inmos®

Reference manual

transputer

inmos

INMOS Limited
PO Box 424
Bristol BS99 7DD
England
Telephone (0272) 290861
Telex 444723

INMOS Corporation
PO Box 16000
Colorado Springs
CO 80935
USA
Telephone (303) 630 4000
TWX 910 920 4904

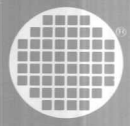
INMOS GmbH
Danziger Strasse 2
8057 Eching
Munich
West Germany
Telephone (089) 319 10 28
Telex 522645

INMOS SARL
Immeuble Monaco
7 rue Le Corbusier
SILIC 219
94518 Rungis Cedex
France
Telephone (1) 687 22 01
Telex 201222

September 1985
72 TRN 006 01

Contents

inmos



Reference manual

transputer architecture

72 TRN 048 01

This manual describes the architecture of the transputer family of products. The first section gives a brief summary of the major features of the transputer architecture and a rationale. The second section gives an overview of the programming model. The third section describes those aspects of the transputer which are common to all members of the transputer family.

Product data manuals, describing the individual products, are combined with this manual.

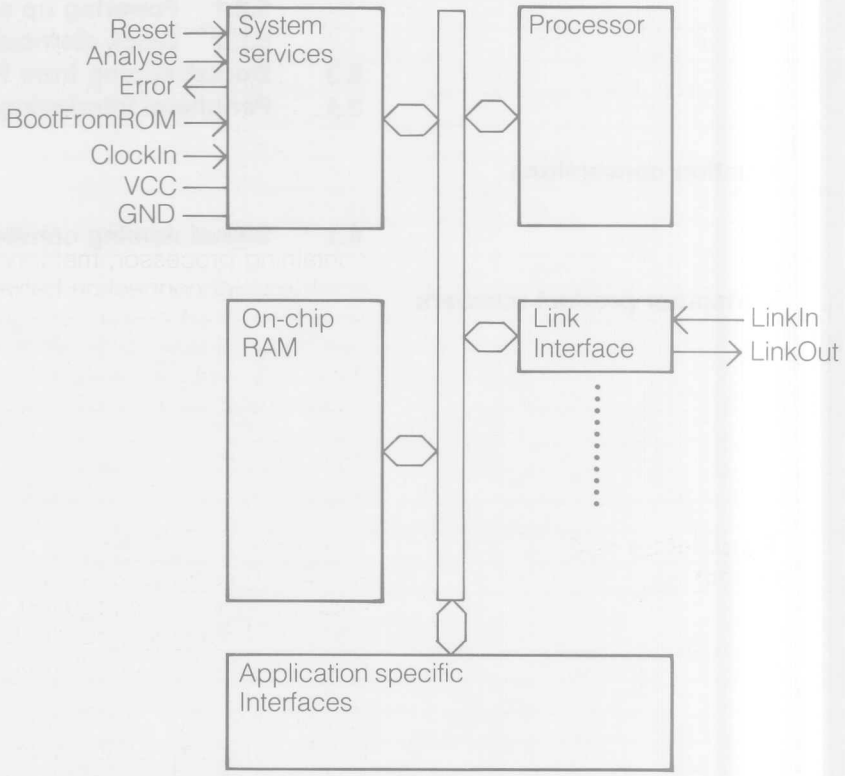
Other information relevant to all transputer products is contained in the occam programming manual (supplied with INMOS software products and available as a separate publication), and the transputer development system manual (supplied with the development system).

The examples given in this manual are outline design studies and are included to illustrate various ways in which transputers can be used. The examples are not intended to provide accurate application designs.

This is the first edition of the manual, dated September 1985.

Contents

Block diagram of a transputer



1.1 Summary

A transputer is a microcomputer with its own local memory and with links for connecting one transputer to another transputer.

The transputer architecture defines a family of programmable VLSI components. The definition of the architecture falls naturally into the *logical* aspects which define how a system of interconnected transputers is designed and programmed, and the *physical* aspects which define how transputers, as VLSI components, are interconnected and controlled.

A typical member of the transputer product family is a single chip containing processor, memory, and communication links which provide point to point connection between transputers. In addition, each transputer product contains special circuitry and interfaces adapting it to a particular use. For example, a peripheral control transputer, such as a graphics or disk controller, has interfaces tailored to the requirements of a specific device.

The transputer can be used as a single chip processor or in networks to build high performance concurrent systems.

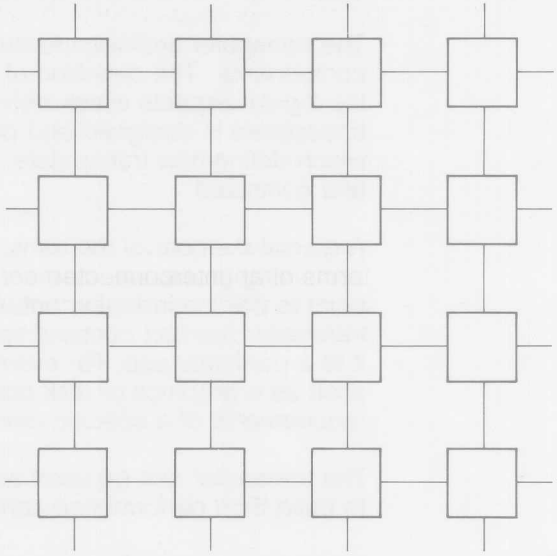
Transputers and occam

The transputer can be programmed in most high level languages, and it has been designed to ensure that compiled programs will be efficient. Where it is required to exploit concurrency, but still to use standard languages, occam can be used as a harness to link modules written in the selected languages.

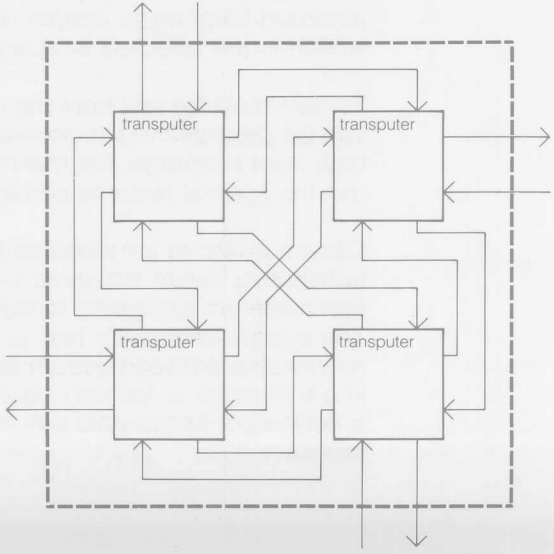
To gain most benefit from the transputer architecture, the whole system can be programmed in occam. This provides all the advantages of a high level language, the maximum program efficiency and the ability to use the special features of the transputer.

Occam provides a framework for designing concurrent systems using transputers in just the same way that boolean algebra provides a framework for designing today's electronic systems from logic gates. The system designer's task is eased because of the architectural relationship between occam and the transputer. A program running in a transputer is formally equivalent to an occam process, so that a network of transputers can be described directly as an occam program.

Transputer array



Four transputer node



1.2.1 System design

The transputer architecture simplifies system design by the use of processes as standard software and hardware building blocks.

The transputer architecture has been designed to satisfy the requirements of system design. An entire system can be designed and programmed in occam, from system configuration down to low level I/O and real time interrupts.

Programming

The software building block is the process. A system is designed in terms of an interconnected set of processes. Each process can be regarded as an independent unit of design. It communicates with other processes along point–to–point channels. Its internal design is hidden, and it is completely specified by the messages it sends and receives. Communication between processes is synchronized, removing the need for any separate synchronisation mechanism.

Internally, each process can be designed as a set of communicating processes. The system design is therefore hierarchically structured. At any level of design, the designer is concerned only with a small and manageable set of processes.

Occam is based on these concepts, and provides the definition of the transputer architecture from the logical point of view (see section 2).

Hardware

Processes can be implemented in hardware. A transputer, executing an occam program, is a hardware process. The process can be independently designed and compiled. Its internal structure is hidden and it communicates and synchronizes with other transputers via its links, which implement occam channels.

Other hardware implementations of the process are possible. For example, a transputer with a different instruction set may be used to provide a different cost/performance trade–off. Alternatively, an implementation of the process may be designed in terms of hard–wired logic for enhanced performance.

The ability to specify a hard–wired function as an occam process provides the architectural framework for transputers with specialized capabilities (e.g., graphics). The required function (e.g., a graphics drawing and display engine) is defined as an occam process, and

implemented in hardware with a standard occam channel interface. It can be simulated by an occam implementation, which in turn can be used to test the application on a development system.

Programmable components

A transputer can be programmed to perform a specialized function, and regarded as a 'black box' thereafter. Some processes can be hard–wired for enhanced performance.

A system, perhaps constructed on a single chip, can be built from combinations of software processes, pre–programmed transputers and hardware processes. Such a system can, itself, be regarded as a component in a larger system.

The architecture has been designed to permit a network of programmable components to have any desired topology, limited only by the number of links on each transputer. The architecture minimizes the constraints on the size of such a system, and the hierarchical structuring provided by occam simplifies the task of system design and programming.

The result is to provide new orders of magnitude of performance for any given application, which can now exploit the concurrency provided by a large number of programmable components.

1.2.2 Systems architecture

Point to point communication links

The transputer architecture simplifies system design by using point to point communication links. Every member of the transputer family has one or more standard links, each of which can be connected to a link of some other component. This allows transputer networks of arbitrary size and topology to be constructed.

Point to point communication links have many advantages over multi–processor buses:

There is no contention for the communication mechanism, regardless of the number of transputers in the system.

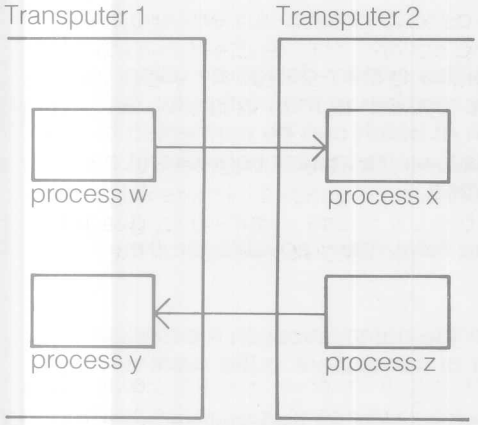
There is no capacitive load penalty as transputers are added to a system.

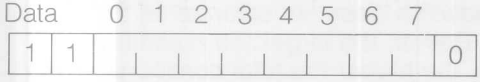
The communications bandwidth does not saturate as the size of the system increases. Rather, the larger the number of transputers in the system, the higher the total communications bandwidth of the system. However large the system, all the connections between transputers can be short and local.

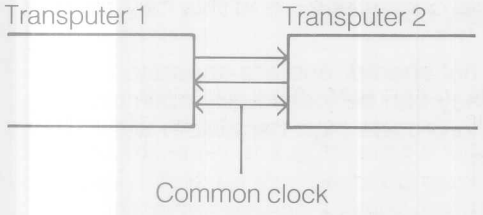
Local memory

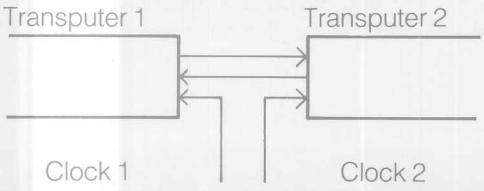
Each transputer in a system uses its own local memory. Overall memory bandwidth is proportional to the number of transputers in the system, in contrast to a large global memory, where the additional processors must share the memory bandwidth.

Because memory interfaces are not shared, and are separate from the communications interfaces, they can be individually optimized on different transputer products to provide high bandwidth with the minimum of external components.









1.2.3 INMOS links

Each link provides two occam channels, one in each direction. This allows an application design, represented in occam, to be directly mapped onto an appropriate network of transputers. The links are designed to make the programming of multi-transputer systems easy. Communications via any link may occur concurrently with communication on all other links and with program execution. Synchronisation of processes at each end of a link is automatic and requires no explicit programming. A program for communication between two processes executing on different transputers is identical to one for two processes executing on the same transputer.

Each link consists of a serial input and a serial output, both of which are used to carry data and link control information. A message is transmitted as a sequence of bytes. After transmitting a data byte, the sending transputer waits until an acknowledge has been received, signifying that the receiving transputer is ready to receive another byte. The receiving transputer can transmit an acknowledge as soon as it starts to receive a data byte so that transmission can be continuous. This protocol synchronizes the communication of each byte of data, ensuring that slow and fast transputers communicate reliably.

The communications protocol is word length independent so that transputers of different word lengths can communicate directly.

The links are designed to make the engineering of transputer systems straightforward. Board layout of two wire connections is easy to design and area efficient. All transputers will support a standard communications frequency of 10 Mbit, regardless of processor performance. Thus transputers of different performance can be directly connected and future transputer systems will directly communicate with those of today.

Link communication is not sensitive to clock phase. Thus, communication can be achieved between independently clocked systems as long as the communications frequency is the same.

The transputer family includes a number of link adaptor devices which provide a means of interfacing transputer links to non–transputer devices.

The programming model for transputers is defined by occam.

The purpose of this section is to describe how to access and control the resources of transputers using occam. Further detail is available in the occam programming manual and the transputer development system manual (provided with the development system).

The transputer development system will enable transputers to be programmed in other industry standard languages. Where it is required to exploit concurrency, but still to use standard languages, occam can be used as a harness to link modules written in the selected languages.

2.1 Overview

In occam, processes are connected to form concurrent systems. Each process can be regarded as a black box with internal state, which can communicate with other processes using point to point communication channels. Processes can be used to represent the behaviour of many things, for example, a logic gate, a microprocessor, a machine tool or an office.

The processes themselves are finite. Each process starts, performs a number of actions and then terminates. An action may be a set of sequential processes performed one after another, as in a conventional programming language, or a set of parallel processes to be performed at the same time as one another. Since a process is itself composed of processes, some of which may be executed in parallel, a process may contain any amount of internal concurrency, and this may change with time as processes start and terminate.

Ultimately, all processes are constructed from three primitive processes – assignment, input and output. An assignment computes the value of an expression and sets a variable to the value. Input and output are used for communicating between processes. A pair of concurrent processes communicate using a one way channel connecting the two processes. One process outputs a message to the channel and the other process inputs the message from the channel.

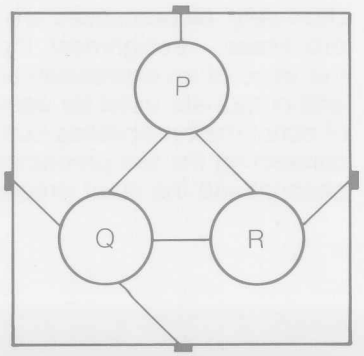
2.1 Overview

The key concept is that communication is synchronized and unbuffered. If a channel is used for input in one process, and output in another, communication takes place when both processes are ready. The inputting and outputting processes then proceed, and the value to be output is copied from the outputting process to the inputting process. Thus communication between processes is like the handshake method of communication used in hardware systems.

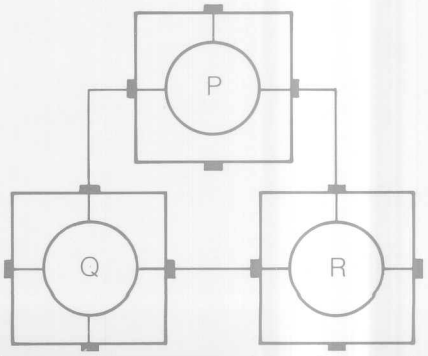
Since a process may have internal concurrency, it may have many input channels and output channels performing communication at the same time.

Every transputer implements the occam concepts of concurrency and communication. As a result, occam can be used to program an individual transputer or to program a network of transputers. When occam is used to program an individual transputer, the transputer shares its time between the concurrent processes and channel communication is implemented by moving data within the memory. When occam is used to program a network of transputers, each transputer executes the process allocated to it. Communication between occam processes on different transputers is implemented directly by transputer links. Thus the same occam program can be implemented on a variety of transputer configurations, with one configuration optimized for cost, another for performance, or another for an appropriate balance of cost and performance.

Mapping processes onto one or several transputers



A program on a single transputer



The same program on three transputers

2 Occam model

2.1 Overview

The transputer and occam were designed together. All transputers include special instructions and hardware to provide maximum performance and optimal implementations of the occam model of concurrency and communications.

All transputer instruction sets are designed to enable simple, direct and efficient compilation of occam. Programming of I/O, interrupts and timing is standard on all transputers and conforms to the occam model.

Different transputer variants may have different instruction sets, depending on the desired balance of cost, performance, internal concurrency and special hardware. The occam level interface will, however, remain standard across all products.

2.2 Occam summary

2.2.1 Processes

After it starts execution, a process performs a number of actions, and then either stops or terminates. Each action may be an assignment, an input, or an output. An assignment changes the value of a variable, an input receives a value from a channel, and an output sends a value to a channel.

At any time between its start and termination, a process may be ready to communicate on one or more of its channels. Each channel provides a one way connection between two concurrent processes; one of the processes may only output to the channel, and the other may only input from it.

Assignment

An assignment is indicated by the symbol `:=`. The example

```
v := e
```

sets the value of the variable `v` to the value of the expression `e` and then terminates. For example, `x := 0` sets `x` to zero, and `x := x + 1` increases the value of `x` by 1.

Input

An input is indicated by the symbol `?` The example

```
c ? x
```

inputs a value from the channel `c`, assigns it to the variable `x` and then terminates.

Output

An output is indicated by the symbol `!` The example

```
c ! e
```

outputs the value of the expression `e` to the channel `c`.

2.2.2 Constructs

A number of processes can be combined to form a construct. A construct is itself a process and can therefore be used as a component of another construct. Each component process of a construct is written two spaces further from the left hand margin, to indicate that it is part of the construct. There are four classes of constructs namely the sequential, parallel, conditional and the alternative construct.

SEQ

A sequential construct is represented by

```
SEQ
  P1
  P2
  P3
  ...
```

The component processes P1, P2, P3 ... are executed one after another. Each component process starts after the previous one terminates and the construct terminates after the last component process terminates. For example

```
SEQ
  c1 ?  x
  x  := x + 1
  c2 !  x
```

inputs a value, adds one to it, and then outputs the result.

Sequential constructs in occam are similar to programs written in conventional programming languages. Note, however, that they provide the performance and efficiency equivalent to that of an assembler for a conventional microprocessor.

PAR

A parallel construct is represented by

```
PAR
  P1
  P2
  P3
  ...
```

The component processes P1, P2, P3 ... are executed together, and are called concurrent processes. The construct terminates after all of the component processes have terminated. For example,

```
PAR
  c1 ? x
  c2 ! y
```

allows the communications on channels c1 and c2 to take place together.

The parallel construct is unique to occam. It provides a straightforward way of writing programs which directly reflect the concurrency inherent in real systems. The implementation of parallelism on a single transputer is highly optimized so as to incur minimal process scheduling overhead.

Communication

Concurrent processes communicate only by using channels, and communication is synchronized. If a channel is used for input in one process, and output in another, communication takes place when both the inputting and the outputting processes are ready. The processes then proceed, and the value to be output is copied from the outputting process to the inputting process.

Communication between processes on a single transputer is via memory–to–memory data transfer. Between processes on different transputers it is via standard links. In either case the occam program is identical.

IF

A conditional construct

```
IF
  condition1
    P1
  condition2
    P2
  ...
```

means that P1 is executed if condition1 is true, otherwise P2 is executed if condition2 is true, and so on. Only one of the processes is executed, and then the construct terminates. For example

```
IF
  x = 0
    y := y + 1
  x <> 0
   SKIP
```

increases y only if the value of x is 0.

ALT

An alternative construct

```
ALT
  input1
    P1
  input2
    P2
  input3
    P3
  ...
```

waits until one of **input1**, **input2**, **input3** ... is ready. If **input1** first becomes ready, **input1** is performed, and then process **P1** is executed. Similarly, if **input2** first becomes ready, **input2** is performed, and then process **P2** is executed. Only one of the inputs is performed, then its corresponding process is executed and then the construct terminates. For example:

```
ALT
  count ? signal
    counter := counter + 1
  total ? signal
    SEQ
      out ! counter
      counter := 0
```

either inputs a signal from the channel **count**, and increases the variable **counter** by 1, or alternatively inputs from the channel **total**, outputs the current value of the counter, then resets it to zero.

The **ALT** construct provides a formal language method of handling external and internal events that must be handled by assembly level interrupt programming in conventional microprocessors.

2.2.3 Repetition

```
WHILE condition
  P
```

repeatedly executes the process **P** until the value of the condition is false. For example

```
WHILE (x - 5) > 0
  x := x - 5
```

leaves **x** holding the value of (**x** remainder 5) if **x** is positive.

2.2.4 Replication

A replicator is used with a constructor to replicate the component process a number of times. For example, a replicator can be used with **SEQ** to provide a conventional loop.

```
SEQ i = 0 FOR n
  P
```

causes the process **P** to be executed **n** times.

A replicator may be used with **PAR** to construct an array of concurrent processes.

```
PAR i = 0 FOR n
  Pi
```

constructs an array of **n** similar processes **P0**, **P1**, ..., **Pn-1**. The index **i** takes the values 0, 1, ..., n–1, in **P0**, **P1**, ..., **Pn-1** respectively.

2.2.5 Types

Every variable, expression and value has a type, which may be a primitive type or an array type. The type defines the length and interpretation of values of the type.

Primitive types

All implementations provide the primitive types.

`CHAN`

Each communication channel provides communication between two concurrent processes.

`TIMER`

Each timer provides a clock which can be used by any number of concurrent processes.

`BOOL`

The values of type **`BOOL`** are true and false.

`BYTE`

The values of type **`BYTE`** are nonnegative numbers less than 256.

`INT`

`INT` is the type of signed integer most efficiently provided by the implementation.

Implementations may also support the following

`INT16`

Signed integers **n** in the range –32768 <= **n** < 32768

`INT32`

Signed integers **n** in the range –2147483648 <= **n** < 2147483648

`INT64`

Signed integers **n** in the range –2**63 <= **n** < 2**63

`REAL32`

Floating point numbers using a sign bit, 8 bit exponent and 23 bit fraction in IEEE Standard P754 draft 10.0 representation

`REAL64`

Floating point numbers using a sign bit, 11 bit exponent and 52 bit fraction in IEEE Standard P754 draft 10.0 representation

2.2.6 Declarations, arrays and subscripts

A declaration `T x :` declares `x` as a new channel, variable, timer or array of type `T`. For example

```
INT x:
P
```

declares `x` as an integer for use in process `P`.

Array types are constructed from component types. For example `[ n ] T` is an array type constructed from `n` components of type `T`.

A component of an array may be selected by subscription, for example `v[e]` selects the `e`'th component of `v`.

A set of components of an array may be selected by subscription, for example `v[FROM e FOR c]` selects the `c` components `v[e], v[e + 1], ... v[e + c - 1]`. A set of components of an array may be assigned, input or output.

2.2.7 Named processes

A process may be given a name. For example

```
PROC square (INT n, INT sqr)
  sqr := n * n :
```

defines the named process `square`.

The name may be used as an abbreviation for the process. For example

```
square (x, sqrx)
```

means

```
sqrx := x * x
```

2.2.8 Expressions

An expression is constructed from the operators given in the table below, variables, numbers, the truth values **TRUE** and **FALSE**, and the brackets **(** and **)**.

Operator	Operand types	Description
+, -, *, /, \	integer, real	arithmetic operators
PLUS, MINUS, TIMES, AFTER	integer	modulo arithmetic
=, <>	any primitive	relational operators
>, <, >=, <=	integer, real	relational operators
AND, OR, NOT	boolean	boolean operators
BITAND, BITOR, >< , BITNOT	integers	bit operators
<<, >>	integer	shift operators

For example, the expression

```
(5 + 7) / 2
```

evaluates to 6, and the expression

```
(#1DF BITAND #F0)>> 4
```

evalutates to **#D** (the character **#** introduces a hexadecimal constant).

A string is represented as a sequence of ASCII characters, enclosed in double quotation marks **"**. If the string has **n** characters, then it is an array of type **[n]BYTE**.

2.2.9 Timer

All transputers incorporate a timer. The implementation directly supports the occam model of time. Each process can have its own independent timer, which can be used for internal measurement or for real time scheduling.

A timer input sets a variable to a value of type **INT** representing the time. The value is derived from a clock, which changes at regular intervals. For example

```
tim ? v
```

sets the variable **v** to the current value of a free running clock, declared as the timer **tim**.

A delayed input takes the following form

```
tim ? AFTER e
```

A delayed input is unable to proceed until the value of the timer satisfies timer **AFTER e**. The comparison performed is a modulo comparison. This provides the effect that, starting at any point in the timer's cycle, the previous half cycle of the timer is considered as being before the current time, and the next half cycle is considered as being after the current time.

2.2.10 Peripheral access

The implementation of occam provides for peripheral access by extending the input and output primitives with a port input/output mechanism. A port is used like an occam channel, but has the effect of transferring information to and from a block of memory locations associated with a peripheral.

Ports behave like occam channels in that only one process may input from a port, and only one process may output to a port. Thus ports provide a secure method of accessing external memory mapped status registers etc.

Note that there is no synchronization mechanism associated with port input and output. Any timing constraints which result from the use of asynchronous external hardware will have to be programmed explicitly. For example, a value read by a port input may depend upon the time at which the input was executed, and inputting at an invalid time would produce unusable data.

During applications development it is recommended that the peripheral is modelled by an occam process connected via channels.

2.3 Configuration

Occam programs may be configured for execution on one or many transputers. The transputer development system provides the necessary tools for correctly distributing a program configured for many transputers.

Configuration does not affect the logical behaviour of a program (see section three, Program development). However, it does enable the program to be arranged to ensure that performance requirements are met.

PLACED PAR

A parallel construct may be configured for a network of transputers by using the **PLACED PAR** construct. Each component process (termed a placement) is executed by a separate transputer. The variables and timers used in a placement must be declared within the placement.

PRI PAR

On any individual transputer, the outermost parallel construct may be configured to prioritize its components. Each process is executed at a separate priority. The first process is the highest priority, the last the lowest. If **P** and **Q** are two concurrent processes with priorities p and q such that $p < q$, then **Q** is only allowed to proceed when **P** cannot proceed.

INMOS standard links

Each link provides one channel in each direction between two transputers.

A channel (which must already have been declared) is associated with a link by a channel association, for example:

```
PLACE LinkOInput AT #80000000
```

In general, programs should be designed to perform all error checking and recovery which is appropriate for the application.

The occam primitive process **STOP** starts but never proceeds. This prevents any process executing it from terminating or engaging in any further communication. An occam process will stop if it encounters an invalid process. The ability for a process to stop is the basic requirement for the design of a redundant system with arbitrary reliability.

A malfunctioning process is handled by taking the following steps:

1 Isolation – the effect of any error is limited to the process; the transputer architecture ensures that no corruption can occur of code and data in another transputer, and the compiler is able to ensure that no corruption can occur of code and data in the same transputer. The offending process stops, preventing it from communicating erroneous data to other processes.

2 Diagnosis – in conjunction with the development system, the states of all processes may be examined.

3 Recovery – continued operation of a system containing an errant subsystem can be ensured by redundancy.

An application may provide its own facilities for (2) and (3) above. The techniques involve the use of facilities specific to the type(s) of transputer(s) being used.

Expressions which cause arithmetic overflow are invalid, and processes which cause array bounds violations are invalid. If the compiler is unable to check that a given construct contains only valid expressions and processes, then extra instructions are compiled to perform the necessary checks at runtime. If the result of the check indicates that an error has occurred, then the processor's **Error** flag is set.

Several courses of action are then possible, depending upon the language being used, its implementation strategy, and how the application is designed to use the specific features of the type(s) of transputer(s) being used.

1 A diagnostic routine or exception handling facility is used to deal with the situation;

2 The process containing the offending instruction stops (the checked occam implementation);

3 Specific facilities are employed which cause the transputer (rather than just a process) to stop on an error. An application may use another transputer to sense the **ErrorOut** pin(s), and then perform diagnosis and/or recovery as required.

The development of programs for multiple processor systems can involve experimentation. In some cases, the most effective configuration is not always clear until a substantial amount of work has been done. For this reason, it is desirable that most of the design and programming can be completed before hardware construction is started.

Logical behaviour

An important property of occam in this context is that it provides a clear notion of 'logical behaviour'; this relates to those aspects of a program not affected by real time effects.

It is guaranteed that the logical behaviour of a program is not altered by the way in which the processes are mapped onto processors, or by the speed of processing and communication. Consequently a program ultimately intended for a network of transputers can be compiled, executed and tested on a single computer used for program development.

Even if the application uses only a single transputer, the program can be designed as a set of concurrent processes which could run on a number of transputers. This design style follows the best traditions of structured programming; the processes operate completely independently on their own variables except where they explicitly interact, via channels. The set of concurrent processes can run on a single transputer or, for a higher performance product, the processes can be partitioned amongst a number of transputers.

It is necessary to ensure, on the development system, that the logical behaviour satisfies the application requirements. The only ways in which one execution of a program can differ from another in functional terms result from dependencies upon input data and the selection of components of an **ALT**. Thus a simple method of ensuring that the application can be distributed to achieve any desired performance is to design the program to behave 'correctly' regardless of input data and **ALT** selection.

4.1 Performance measurement

Performance information is useful to gauge overall throughput of an application, and has to be considered carefully in applications with real time constraints.

Prior to running in the target environment, an occam program should be relatively mature, and indeed should be correct except for interactions which do not obey the occam synchronization rules. These are precisely the external interactions of the program where the world will not wait to communicate with an occam process which is not ready. Thus the set of interactions that need to be tested within the target environment are well identified.

Because, in occam, every program is a process, it is extremely easy to add monitor processes or simulation processes to represent parts of the real time environment, and then to simulate and monitor the anticipated real time interactions. The occam concept of time and its implementation in the transputer is important. Every process can have an independent timer enabling, for example, all the real time interactions to be modelled by separate processes and any time dependent features to be simulated.

4.2 Separate compilation of occam and other languages

A program portion which is separately compiled, and possibly written in a language other than occam, may be executed on a single transputer.

If the program is written in occam, then it takes the form of a single `PROC`, with only channel parameters. If the program is written in a language other than occam, then a run–time system is provided which provides input/output to occam channels.

Such separately compiled program portions are linked together by a framework of channels, termed a harness. The harness is written in occam. It includes all configuration information, and in particular specifies the transputer configuration in which the separately compiled program portion is executed.

Transputers are designed to allow efficient implementations of high level languages, such as C, Pascal and Fortran. Such languages will be available in addition to occam.

At runtime, a program written in such a language is treated as a single sequential occam process. Facilities are provided in the implementations of these languages to allow such a program to communicate on 'occam' channels. It can thus communicate with other such programs, or with programs written in occam. These programs may reside on the same transputer, in which case the channels are implemented in store, or may reside on different transputers, in which case the channels are implemented by transputer links.

It is therefore possible to implement 'occam' processes in conventional high level languages, and arrange for them to communicate. It is possible for different parts of the same application to be implemented in different high level languages.

The standard input and output facilities provided within these languages are implemented by a well–defined protocol of communications on 'occam' channels.

The development system provides facilities for management of separately compiled occam.

4.3 Memory map and placement

The low level memory model is of a signed address space.

Memory is byte addressed, the lowest addressed byte occupying the least significant byte position within the word.

The implementation of occam supports the allocation of the code and data areas of an occam process to specific areas of memory. Such a process must be a separately compiled **PROC**, and must not reference any variables and timers other than those declared within it.

No location may be allocated more than once.

5.1 INMOS serial links

5.1.1 Summary

All transputers have several links. A link implements a pair of occam channels, one in each direction.

The link protocol and electrical characteristics form a standard for all INMOS transputer and peripheral products.

All transputers support a standard link communications frequency of 10 megabits per second. Some devices also support faster data rates. Maintaining a standard communications frequency means that devices of mixed performance and vintage can intercommunicate easily.

Each link consists of two unidirectional signal wires carrying both data and control bits. The link signals are TTL compatible so that their range can be easily extended by inserting buffers.

The INMOS communication links provide for communication between devices on the same printed circuit board or between printed circuit boards via a back plane. They are intended to be used in electrically quiet environments in the same way as logic signals between TTL gates.

Each link interface is autonomous and has an output and an input signal, both of which are used to carry data and protocol bits. A message is transmitted as a sequence of bytes. After transmitting a data byte, the sending transputer waits until an acknowledge has been received, signifying that the receiving transputer is ready to receive another byte, before transmitting the next byte. The protocol permits the receiving transputer to transmit an acknowledge as soon as it starts to receive a data byte.

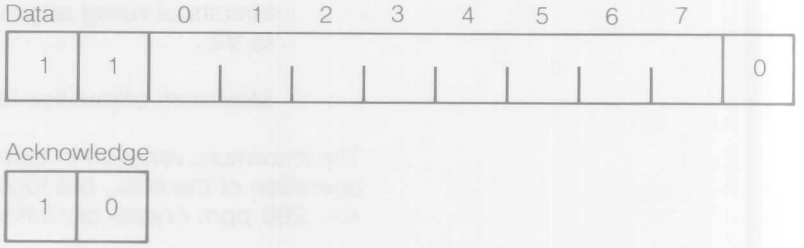
Each link implements two occam channels, and the protocol provides end–to–end channel synchronization.

The number of links, and any communication speeds in addition to the standard speed of twice the input clock frequency, are given in the product data for each product.

5.1.2 Link protocol

Each unidirectional link signal carries both data and control information. A message is transmitted as a sequence of data packets. Each data packet is transmitted as a one bit followed by a further one bit followed by the eight data bits followed by a zero bit. After transmitting a data packet, the sender waits until an acknowledge packet is received; this consists of a one bit followed by a zero bit.

Link protocol



The quiescent state of the link signals is low, for a zero.

Data and acknowledge packets are multiplexed down each signal line. The protocol allows for an acknowledge to be transmitted as soon as the data packet has been identified, provided that there is sufficient buffer space for another data packet and the inputting process is ready to receive the previous data packet.

5.1.3 Electrical specification of links

The link input signals and output signals are standard TTL compatible signals.

For correct functioning of the links the following specifications must be met:

Maximum difference in clock frequency between two transputers joined by a link is 400 ppm.

Link input rising edges must be monotonic within the range VIH to VIL.

Maximum capacitive load is 50 pf.

The maximum variation in clock frequency is tight enough for correct operation of the links, but loose enough to allow the use of low cost +/– 200 ppm crystal oscillators.

Each transputer product also has specified the maximum permissible skew in buffering between the rising edge delay and the falling edge delay. This specification is given in the product data for each product.

Provided that this specification is met then any buffering employed may introduce an arbitrary delay into a link signal without affecting its correct operation.

5.2 System services

5.2.1 Powering up and down, running and stopping

At all times the specification of input voltages with respect to the GND pins must be met. This includes the times when the VCC pins are ramping to 5 V, and also while they are ramping from 5 V down to 0 V.

The system services comprise the clocks, power, and signals used for initialization.

The specification includes minimum times that VCC must be within specification, the input clock must be oscillating, and the Reset signal must be high before Reset goes low. These specifications ensure that internal clocks and logic have settled before the transputer starts.

When Reset goes low the transputer initializes the memory interface (if present and configurable).

The processor and INMOS serial links start after the memory interface has been initialized. The transputer obeys a bootstrap program which can either be in off–chip ROM or can be received from one of the links. How to specify where the bootstrap program is taken from depends upon the type of transputer being used. The program will normally load up a larger program either from ROM or from a peripheral such as a disk.

Normal running continues with power switched on until the system is switched off. During power down, as during power up, the input and output pins must remain within specification with respect to both GND and VCC.

A software error, such as arithmetic overflow, array bounds violation or divide by zero, causes an error flag to be set in the transputer processor. The flag is directly connected to the Error pin. Both the flag and the pin can be ignored, or the transputer stopped, using implementation dependent mechanisms. Stopping the transputer on an error means that the error cannot cause further corruption.

As well as containing the error in this way it is possible to determine the state of the transputer and its memory at the time the error occurred, using implementation dependent mechanisms.

5.2.2 Clock distribution

Transputers use a low frequency input clock and generate internal clocks from it to avoid most of the problems of distributing high frequency clocks. Within limits the mark–to–space ratio, the voltage levels and the transition times are immaterial. The asynchronous data reception of the links means that the clock phase between chips does not matter.

The important characteristic of the transputer's input clock is its stability, such as is provided by a crystal oscillator. An R–C oscillator is inadequate. The edges of the clock should be monotonic (without kinks), and should not undershoot below –0.5 V.

5.3 Bootstrapping from ROM or from a link

The program which is executed after reset can either come from ROM in the transputer's address space or it can come through any one of the transputer's INMOS serial links.

The transputer bootstraps from ROM by transferring control to the location below the top byte of memory.

If bootstrapping from a link, the transputer bootstraps from the first link to receive a message. The first byte of the message is the count of the number of bytes of program which follow. The program is loaded into memory starting at an implementation dependent location **MemStart**, and then control is transferred to this address.

Messages subsequently arriving on other links are not acknowledged until the transputer processor obeys a program which inputs from them.

The load path for a network of transputers is derived by the transputer development system, which ensures that the first message each transputer receives is the bootstrap program.

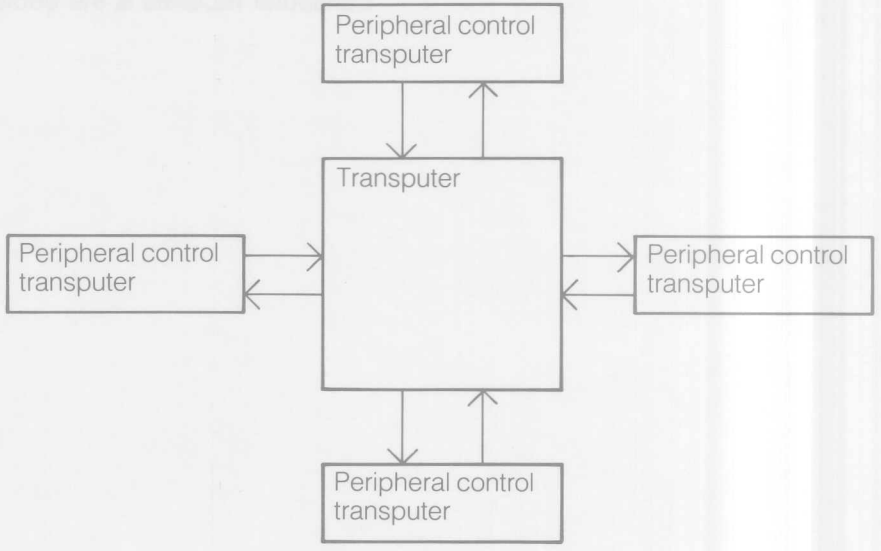
5.4 Peripheral interfacing

All transputers contain one or more INMOS serial links. Certain transputer products also have other application–specific interfaces. The peripheral control transputers contain specialized interfaces to control a specific peripheral or peripheral family.

In general, a transputer based application will comprise a number of transputers which communicate using INMOS links. There are three methods of communicating with peripherals.

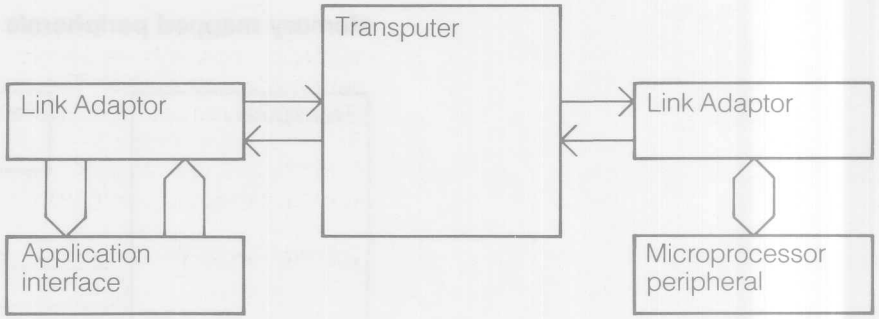
The first is by employing peripheral control transputers (eg for graphics or disks), in which the transputer chip connects directly to the peripheral concerned. The interface to the peripheral is implemented by special purpose hardware within the transputer. The application sofware in the transputer is implemented as an occam process, and controls the interface via occam channels linking the processor to the special purpose hardware.

Transputer with peripheral control transputers



The second method is by employing link adaptors. These devices convert between a link and a specialized interface. The link adaptor is connected to the link of an appropriate transputer, which contains the application designer's peripheral device handler implemented as an occam process.

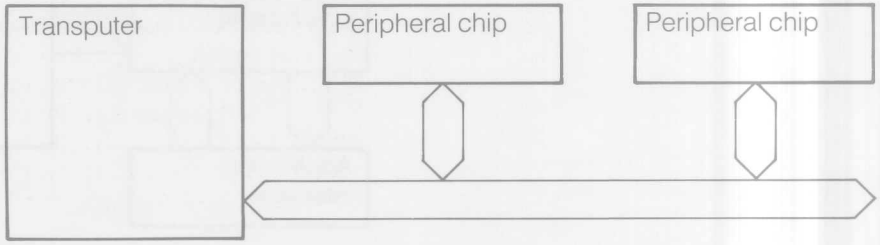
Transputer with link adaptors



The third method is by memory mapping the peripheral onto the memory bus of a transputer. The peripheral is controlled by memory accesses issued as a result of **PORT** inputs and outputs. The application designer's peripheral device handler provides a standard occam channel interface to the rest of the application.

The first transputers implement an event pin which provides a simple means for an external peripheral to request attention from a transputer.

Memory mapped peripherals



In all three methods, the peripheral driver interfaces to the rest of the application via occam channels. Consequently, a peripheral device can be simulated by an occam process. This enables testing of all aspects of a transputer system before the construction of hardware.

The transputer architecture is 'little endian'.

The bits in a byte are numbered 0 to 7, with bit 0 the least significant. The bytes in words are numbered from 0, with byte 0 least significant. In general, wherever a value is treated as a number of component values, the components are numbered in order of increasing numerical significance, with the least significant component numbered 0. Where values are stored in memory, the least significant component value is stored at the lowest (most negative) address. Similarly, components of arrays are numbered starting from 0, and stored in memory with component 0 at the lowest address.

Where a byte is transmitted serially, it is always transmitted least significat bit (0) first. In general, wherever a value is transmitted as a number of component values, the least significant component is transmitted first. Where an array is transmitted serially, component 0 is transmitted first. Consequently, block transfers to and from memory are performed starting with the lowest (most negative) address and ending with the highest (most positive) one.

In diagrams, the least significant component of a value is to the right hand side of the diagram, component 0 of an array is at the bottom of the diagram and memory locations with more negative addresses are also to the bottom of the diagram.

6.1 Signal naming conventions

The signal names, identifying the individual pins on a transputer chip, have been chosen to avoid being cryptic, giving as much information as possible.

All transputer signals described in the text of this manual are printed in **bold**.

The majority of transputer signals are active high. Those which are active low have names commencing with **not**.

All INMOS products – both memories and transputers – have a number of the form

IMS xxxx–xx

The main field identifies the product, and the field after the hyphen is used for speed variants, etc. Extra letters are sometimes introduced, eg for military quality products.

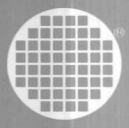
The initial character of the main field is a digit for memory products, a letter for transputer products. The particular letter indicates the type of transputer product:

IMS Cxxx	Communications adaptors
IMS Gxxx	Graphics transputers
IMS Mxxx	Mass storage transputers
IMS Txxx	Transputers

Support products are numbered as follows:

IMS Bxxx	Board level product
IMS Dxxx	Development system
IMS Lxxx	Literature
IMS Pxxx	Occam programming system
IMS Sxxx	Software product

inmos



Product data

IMS T414 transputer

72 TRN 049 01

The IMS T414 is the first in a family of transputers, all of which are consistent with the INMOS transputer architecture, described in the transputer architecture manual.

This manual details the product specific aspects of the IMS T414 and contains all the data necessary to engineer and program the device.

Other information relevant to all transputer products is contained in the occam programming manual (supplied with INMOS software products and available as a separate publication), and the transputer development system manual (supplied with the development system).

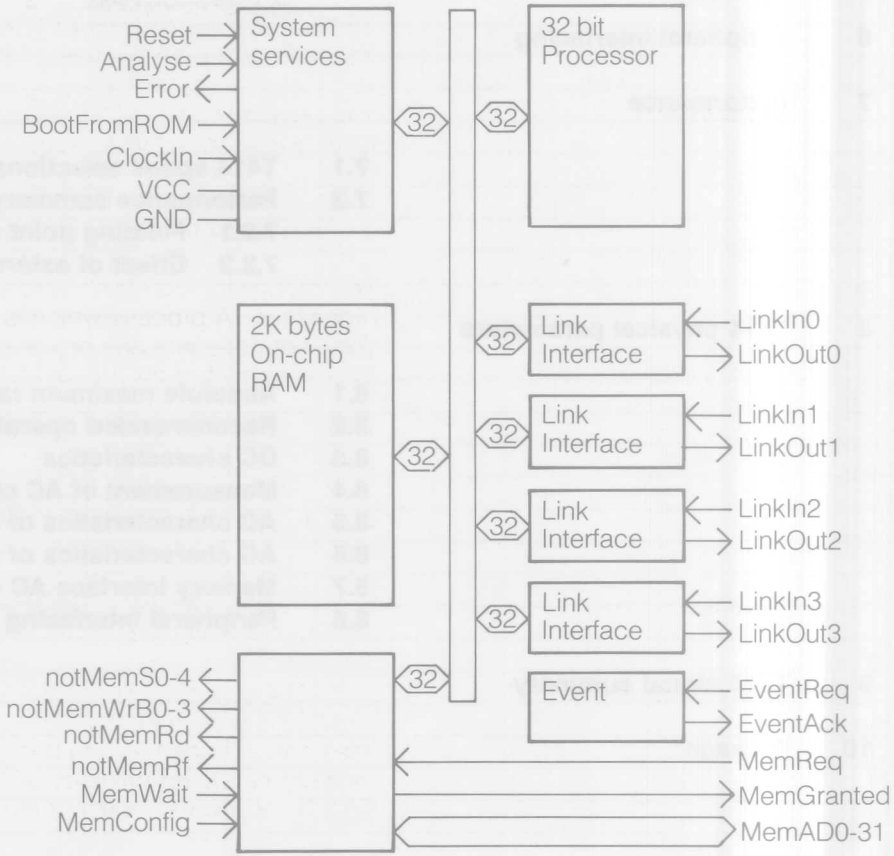
The examples given in this manual are outline design studies and are included to illustrate various ways in which transputers can be used. The examples are not intended to provide accurate application designs.

This is the first edition of the manual, dated September 1985.

Contents

Contents

T414 block diagram



IMS T414

The IMS T414 integrates a 32–bit microprocessor, four standard transputer communications links, 2K bytes of on–chip RAM, a memory interface and peripheral interfacing on a single 1.5 micron CMOS chip.

Processor

The design of the T414 achieves compact programs, efficient high level language implementation and provides direct support for the occam model of concurrency. Procedure calls, process switching and interrupt latency are all sub–microsecond. The T414 provides high performance arithmetic and direct support for floating point operations.

The processor shares its time between any number of concurrent processes. A process which is waiting for communication or a timeout does not consume any processor time. Two levels of process priority enable fast interrupt response to be achieved.

Links

The T414 uses a DMA block transfer mechanism to transfer messages between its memory and another transputer product via the INMOS links. The link interfaces and the processor all operate concurrently, allowing the processing to continue while the data is being transferred on all of the links.

Memory

The 2K bytes of RAM provides a maximum data rate of 80 Mbytes/s with multiport access for the processor and links.

Memory interface

The T414 can directly access a linear address space up to 4 Gbytes. The 32 bit wide memory interface uses mutiplexed data and address lines and provides a data rate of up to 25 MBytes/s. A configurable memory controller provides all timing, control and DRAM refresh signals for a wide variety of mixed memory systems.

Peripheral interface

The memory controller supports memory mapped peripherals, which may use DMA. Links may be interfaced to peripherals via an INMOS link adapter. A peripheral can request attention via the event pin.

Time

The processor includes timers for both high and low priority processes.

Error handling

High–level language execution is made secure with array bounds checking, arithmetic overflow detection etc. A flag is set when an error is detected. The error can be handled internally by software or externally by sensing the error pin. System state is preserved for subsequent analysis.

The optimal method of programming the T414 processor is to use occam. See the occam programming manual for full details.

The processor provides direct support for the occam model of concurrency and communication. It has a scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. The number of registers which hold the process context is small and this, combined with fast on chip RAM, provides a sub–microsecond process switch time.

Process communication is implemented by memory to memory block move operations. These fully utilize the bandwidth available from the on–chip RAM.

2.1 T414 types

The implementation of occam for the T414 transputer supports the following types:

CHAN	Each communication channel provides communication between two concurrent processes.
TIMER	Each timer provides a clock which can be used by any number of concurrent processes.
BOOL	The values of type **BOOL** are true and false.
BYTE	The values of type **BYTE** are nonnegative numbers less than 256.
INT	Signed integers **n** in the range –2147483648 <= **n** < 2147483648
INT16	Signed integers **n** in the range –32768 <= **n** < 32768
INT32	Signed integers **n** in the range –2147483648 <= **n** < 2147483648
INT64	Signed integers **n** in the range –2**63 <= **n** < 2**63
REAL32	Floating point numbers using a sign bit, 8 bit exponent and 23 bit fraction in IEEE Standard P754 draft 10.0 representation
REAL64	Floating point numbers using a sign bit, 11 bit exponent and 52 bit fraction in IEEE Standard P754 draft 10.0 representation

2.2 T414 process multiplexing

The T414 supports two levels of priority – in occam notation, a `PRI PAR` (priority parallel) process may have two components. The priority 1 (low priority) processes are executed whenever there are no active priority 0 (high priority) processes.

High priority processes

High priority processes are expected to execute for a short time. If one or more high priority processes are able to proceed, then one is selected and runs until it has to wait for a communication, a timer input, or until it completes processing.

Low priority processes

If no process at high priority is able to proceed, but one or more processes at low priority are able to proceed, then one is selected.

Low priority processes are periodically timesliced to provide an even distribution of processor time between computationally intensive tasks.

If there are **n** low priority processes, then the maximum latency from the time at which a low priority process becomes active to the time when its starts processing is 2**n** – 2 timeslice periods. It is then able to execute for between one and two timeslice periods, less any time taken by high priority processes.

Each timeslice period lasts for 4096 cycles of the input clock **ClockIn** (approximately 800 microseconds at the standard frequency of 5 MHz).

To ensure that each low priority process proceeds, high priority processes should never occupy the processor continuously for a period of time equal to a timeslice period. A good guideline is to ensure that, if there are a total of **n** high priority processes, then each limits its activity to much less than 1/**n**'th of any 800 microsecond period.

Interrupt latency

If a high priority process is waiting for an external channel to become ready, and if there are no active high priority processes, then the interrupt latency (from when the channel becomes ready to when the process starts executing) is typically 12 processor cycles, maximum 58 cycles (assuming use of on chip RAM).

2.3 T414 Error flag

Expressions which cause arithmetic overflow are invalid, and processes which cause array bounds violations are invalid. If the compiler is unable to check that a given construct contains only valid expressions and processes, then extra instructions are compiled to perform the necessary checks at runtime. If the result of the check indicates that an error has occurred, then the processor's **Error** flag is set.

In the implementation of occam, the offending process stops when the **Error** flag is set. The T414 can be initialized so that the processor halts when the **Error** flag is set. The appropriate initialization sequence is provided by the development system.

If the processor has been halted as the result of an error, the links continue with any outstanding transfers, the memory interface continues to provide refresh cycles, and the transputer may be analysed, see section 3.2.

When a high priority process pre–empts a low priority process, the current value of the **Error** flag is preserved, and the **Error** flag is reset. When, finally, there are no high priority processes able to run, the current state of the **Error** flag is lost, and the preserved state is restored as part of commencing to execute the pre–empted low priority process.

This ensures that the state of the **Error** flag is preserved during the evaluation of an expression or a sequence of assignments.

2.4 T414 memory map

The address space of the T414 is signed and byte addressed. Words are, by convention, aligned on four–byte boundaries.

Addresses in the range `[#80000000 FOR #800]` reference on chip memory.

The first 18 word locations (72 bytes) of the address space are used for system purposes. The next available location is, by convention, referred to as `MemStart`. A suitable definition of `MemStart` for incorporation into an occam program is:

```
VAL MemStart IS #80000048 :
```

The top of address space is used, by convention, for ROM based code. If the transputer is configured to bootstrap from ROM, then the processor commences execution from address `#7FFFFFFE`. If the transputer is configured to use an externally defined memory interface configuration, then this is stored at locations `#7FFFFF6C` to `#7FFFFFF8`.

2.5 T414 timer

At the standard **ClockIn** cycle rate of 5MHz, the high priority timer has a resolution of one microsecond and cycles approximately every 4,295 seconds (approximately 71 minutes). When comparing timer values, it should be noted that each half cycle is approximately 35 minutes.

The low priority timer has a resolution of 64 microseconds, and each half cycle is approximately 37 hours.

2.6 T414 event pins

An attention–seeking peripheral may signal the T414 via the EventReq pin, which the T414 handshakes using EventAck. The T414 implements a hardware channel to allow a low to high transition on the EventReq pin to be communicated to a process as a synchronizing message.

An occam channel (which must already have been declared) may be associated with EventReq pin by a channel association. The conventional name and the value used for this channel are given by

```
PLACE Event AT #80000020 :
```

Note that occam and other compilers may identify channels using small integers, avoiding the need to specify addresses. Full details are given in the reference manual provided with the development system for details. `Event` behaves like an ordinary occam channel, and a process may synchronize with a low to high transition on the EventReq pin by using the occam construct:

```
Event ? signal
```

The process waits until the channel `Event` is ready. The channel is made ready by the transition on the EventReq pin (this may occur before the process attempts to input).

When the process is able to proceed, and if it executes at high priority, then it will take priority over any low priority process which may be executing when the transition occurs on the EventReq pin.

2.7 T414 link associations

The conventional names and the values used for these channels are

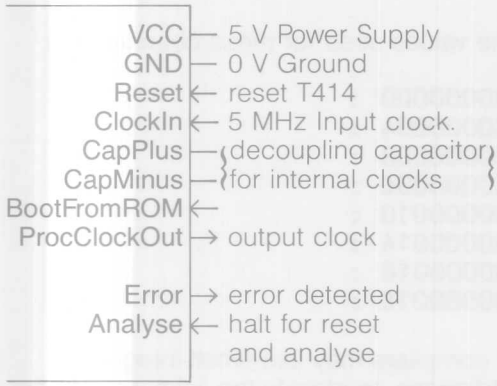
```
PLACE Link0Output AT #80000000 :
PLACE Link1Output AT #80000004 :
PLACE Link2Output AT #80000008 :
PLACE Link3Output AT #8000000C :
PLACE Link0Input  AT #80000010 :
PLACE Link1Input  AT #80000014 :
PLACE Link2Input  AT #80000018 :
PLACE Link3Input  AT #8000001C :
```

Note that the occam and other compilers may use small integer offsets, rather than absolute addresses, to identify the links, so that the same **PLACE** statements may be use for transputers of different word lengths. Full details are given in the reference manual provided with the development system.

2.8 T414 arithmetic procedures

A set of predefined named processes (**PROC**s) is provided to support the efficient implementation of multiple length and floating point arithmetic. In the following table, **n** gives the number of places shifted. Full details of these procedures are provided in the reference manual supplied as part of the development system.

PROC		Cycles	+Cycles for Parameter Access
LongAdd		2	7
LongSum		3	8
LongSub		2	7
LongDiff		3	8
LongProd		34	8
LongDiv		36	8
ShiftRight	(n<32)	4+n	8
	(n>31)	n–25	8
ShiftLeft	(n<32)	4+n	8
	(n>31)	n–24	8
Normalise	(n<32)	n+7	7
	(n>31)	n–23	7
	(n=64)	4	7



The system services comprise the clocks, power and initialization used by the whole of the transputer. The **Reset** and **Analyse** inputs enable the T414 to be initialized, for example on power up, or halted in a way which preserves its state for subsequent analysis. Whilst the T414 is running both **Reset** and **Analyse** are held low. The **Error** signal is directly connected to the processor's **Error** flag (see section 2.3).

3.1 Reset

The T414 is initialized by pulsing **Reset** high whilst holding **Analyse** low. Operation ceases immediately and all state information is lost. When **Reset** goes low the transputer sets up the memory interface configuration as described in section 5.6.

The processor and links start operating after the memory interface configuration cycle is complete and sufficient refresh cycles have been executed to initialize any dynamic RAM.

The processor then bootstraps. If the **BootFromROM** input is high it will start to execute code starting from address #7FFFFFFE. If **BootFromROM** is low it will load a program from a link and then execute it.

When initialising following power–on, a time is specified during which **VCC** must be within specification, **Reset** must be high, and the input on **ClockIn** must be oscillating. **Reset** is taken low after the specified time has elapsed.

3.2 Analyse

A system built from transputers may be brought to a halt in a consistent state which is preserved for subsequent analysis. This analysis is performed in a manner similar to bootstrapping. The transputer development system includes appropriate bootstrap and analysis software.

A single signal may be applied to the **Analyse** inputs of all the transputers in the system. The system is analysed by first taking **Analyse** high. This causes each T414 to halt, after a short period of time, in a consistent internal state. When the system has halted, **Reset** is taken high for the specified hold time, after which it is taken low. **Analyse** is then taken low, at which time each T414 bootstraps.

When **Analyse** is taken high, the processor will halt within three timeslice periods (approximately three milliseconds), plus the time taken for any priority process to cease processing. Any outputting links continue until they complete the remainder of the current word. Input links will continue to receive data. Provided that there are no delays in sending acknowledgements, the links in a system will therefore cease activity within a few microseconds. Sufficient time must be allowed to allow the processor to halt and link traffic to cease before **Reset** is asserted.

The system must be designed so that links connected to the external world are quiet during reset.

The memory interface is not affected by **Analyse**, or by **Reset** while **Analyse** is held high. If refresh cycles are enabled, it continues to refresh external dynamic RAM.

3.3 Bootstrapping

3.3.1 Bootstrapping from ROM

To bootstrap from ROM, the BootFromROM input is wired to VCC.

The T414 bootstraps from ROM by executing a process at low priority. Control is transferred to the byte location one below the top of memory (#7FFFFFFE), and `Memstart` (#80000048) is used as the location of the process workspace.

3.3.2 Bootstrapping from a link

To bootstrap from a link, the BootFromROM input is wired to GND.

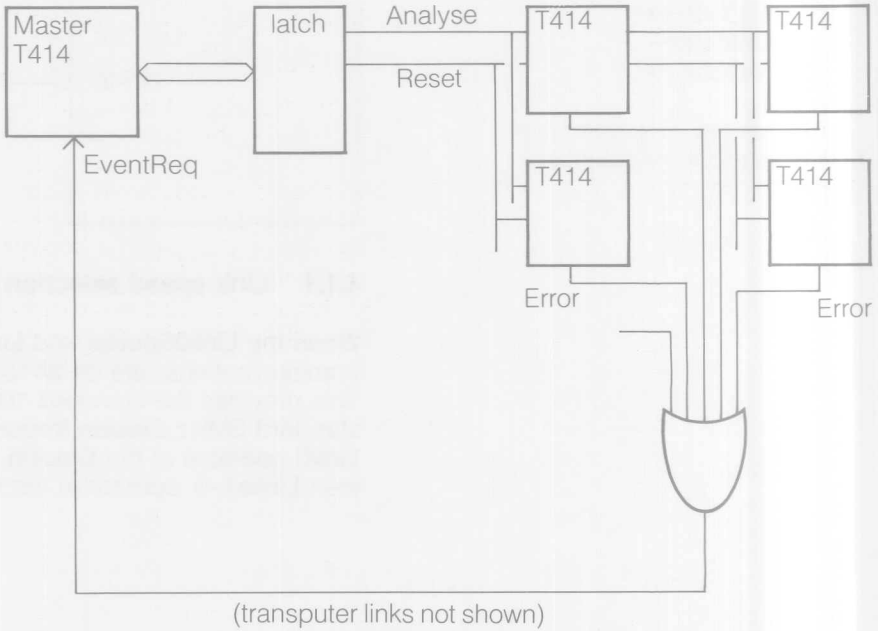
The T414 bootstraps from a link by waiting for the first byte (the control byte) to arrive on any of the four link inputs. The link providing this byte is denoted the bootstrap link.

The value of the control byte must be two, or greater, and is a count of the number of bytes to be input. The T414 then inputs the bytes into memory, starting at `MemStart` (#80000048). The T414 then commences to execute a process at low priority by transferring control to `Memstart`, and by using the first available word of memory above the loaded code as the location of the process workspace.

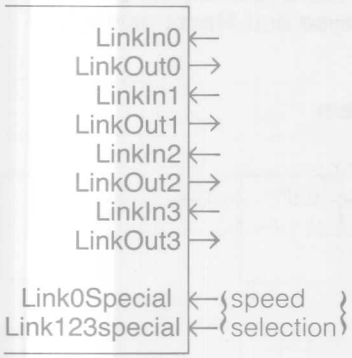
3.4 Using Error and Analyse

Analyse may be used in conjunction with the **Error** output to isolate errors in a multi–transputer system. A transputer which has signalled an error may be halted and subsequently analysed under the control of a 'master' transputer. This could, for example, be achieved by connecting the 'OR' of all the **Error** pins to the **EventReq** pin on the master transputer, and connecting the **Analyse** and **Reset** pins to the master transputer as a 'peripheral'.

Error treatment in multi–transputer system



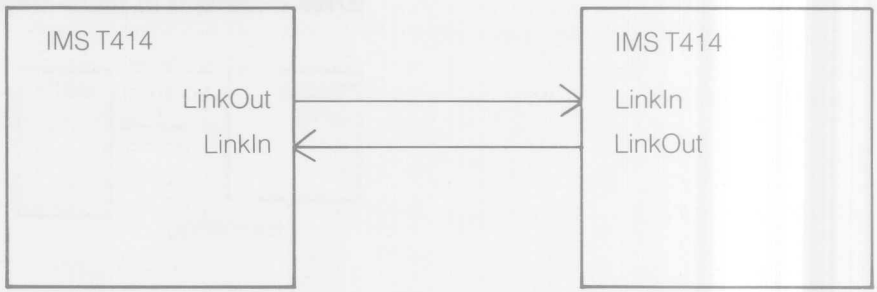
4.1 Standard transputer links



The T414 provides four standard links, each providing two unidirectional point–to–point occam channels.

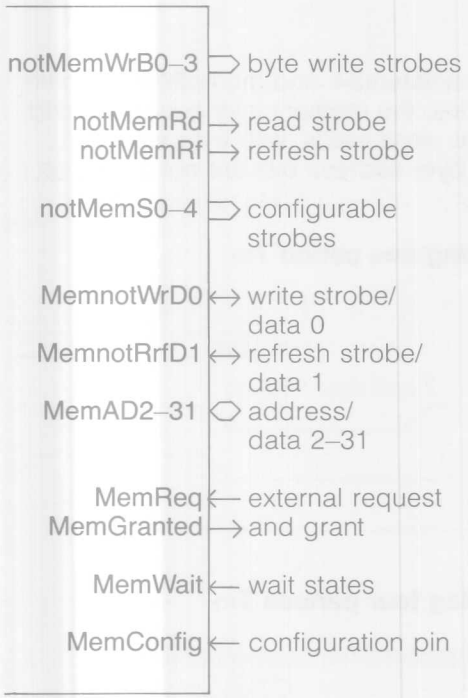
The T414 links implement the standard inter transputer communications protocol. The links are connected by wiring a **LinkOut** to a **LinkIn**.

Link connection



4.1.1 Link speed selection

When the **Link0Special** and **Link123Special** inputs are held low the communications rate on all four links is twice the input clock frequency. This provides the standard 10Mbits/sec communications rate at the standard 5MHz **ClockIn** frequency. If **Link0Special** is wired high then **Link0** operates at the **ClockIn** frequency. If **L123Special** is wired high then **Links1–3** operate at the **ClockIn** frequency.



The memory interface uses a 32–bit wide address/data bus and its controller may be configured, on reset, to suit a wide variety of different memory systems. One of 13 preset configurations may be selected or the configuration may be supplied externally.

The T414 memory interface cycle has six states, referred to as 'Tstates'. The duration of each Tstate is chosen to suit the memory used.

Tstate	
T1	address setup time before address valid strobe
T2	address hold time after address valid strobe
T3	read cycle tristate/write cycle data setup
T4	extended for wait states
T5	read or write data
T6	end tristate/data hold

Each Tstate is configured to have a duration of one to four periods **Tm**. One period **Tm** is half the processor cycle time. In addition, **T4** may be extended by wait states.

Separate write strobes **notMemWrB0**, **notMemWrB1**, **notMemWrB2**, **notMemWrB3**, are provided for each byte. The read strobe, **notMemRd**, ensures the bus is tristated before read data is enabled.

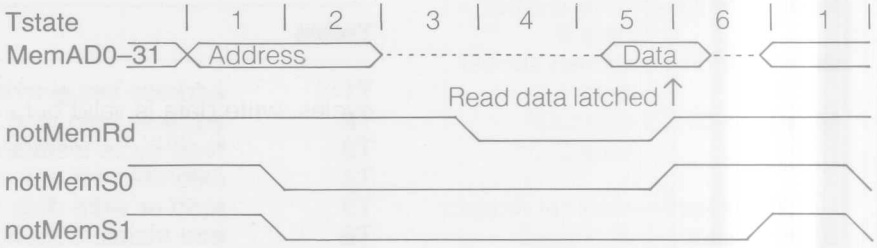
The use of the strobes **notMemS0** to **notMemS4** will depend upon the memory system, for example they may be used directly to control dynamic RAM. The durations of **notMemS1** to **notMemS4** are denoted respectively as **s1** to **s4**, and are independently configurable to be between 1 and 31 periods **Tm**.

Signal	Starts	Ends
notMemS0	T2	T6
notMemS1	T2	T2+(Tm*s1)
notMemS2	T2+(Tm*s2)	T6
notMemS3	T2+(Tm*s3)	T6
notMemS4	T2+(Tm*s4)	T6

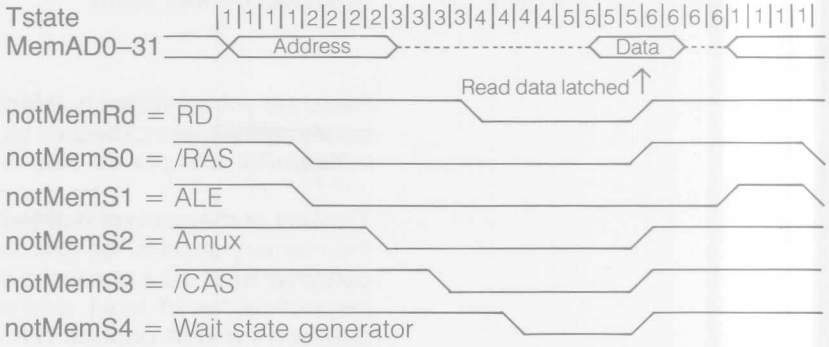
5.2 Read cycles

Read cycles generate **notMemS0–notMemS4** and the **notMemRd** gate data signal. Input data is latched into the memory interface at the end of **T5**. Byte reads are performed as word reads, with byte selection performed by the processor. The byte address bits are not output on the **MemAD** signals.

Read cycle with each Tstate lasting one period Tm



Read cycle with each Tstate lasting four periods Tm



5.3 Write cycles

Write cycles generate **notMemS0–notMemS4** and the **notMemWrB0–3** write data strobe signals. A separate **notMemWrB** signal is provided for each byte. A word write uses all the **notMemWrB** signals. If a particular byte is not to be written, then the corresponding data outputs are tristated.

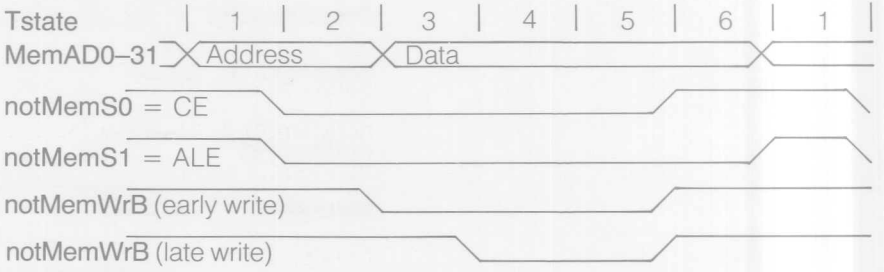
Early indication of a write cycle is provided by a low signal on **MemnotWrD0** during **T1** and **T2**, with the same timing as the address signals.

Write cycles may be configured to be either early or late. In late write cycles, write data is valid before the leading edge of **notMemWrB**, and held valid until after the trailing edge of **notMemWrB**.

Early write cycles provide a wide **notMemWrB** pulse, with data valid at the trailing edge of **notMemWrB**.

The **notMemRd** signal is held high during write cycles.

Write cycles, early and late



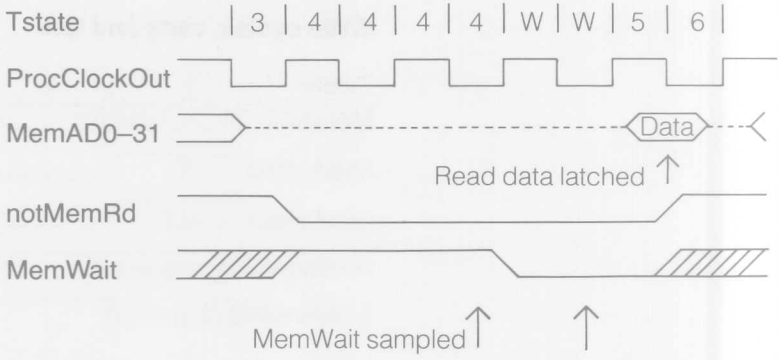
5.4 Use of the MemWait input

An interface cycle may be extended by holding the **MemWait** input high. This may be done by connecting **MemWait** to one of the configurable strobes, **notMemS1** to **notMemS4**, or to an external signal. The **MemWait** input is sampled during the two **Tm** periods before the end of **T4**. If it is high, then **T4** is extended by two **Tm** periods. During this time, **MemWait** is sampled again and, if high, two more **Tm** periods are inserted. If low, the cycle continues with **T5**.

The sample is synchronous and occurs whilst **ProcClockOut** is low. If a strobe, eg **notMemS4** is used to extend the cycle, the strobe must be taken low an even number of periods **Tm** after the start of the memory interface cycle.

Note that if **MemWait** is used to extend cycles for longer than the interval between refresh cycles, refresh cycles are liable to be lost.

Cycle configured for long T4**, with** MemWait **extending the cycle by two periods.**



5.5 Refresh

Refresh can be configured to be enabled or disabled. If Refresh is enabled, the interval between refresh cycles can be configured to be 18, 36, 54 or 72 periods of the input clock signal **ClockIn**.

Refresh interval in periods of ClockIn	**Interval with** ClockIn **frequency of 5.0 MHz** (in microseconds)	**Configuration encoding**
18	3.6	00
36	7.2	01
54	10.8	10
72	14.4	11

Refresh cycles generate all the strobes **notMemS1** to **notMemS4**, but neither **notMemRd** nor the **notMemWrB** signals. **notMemRf** goes low before **T1** and remains low until the end of **T6**. Refresh is also indicated by **MemnotRfD1** going low during **T1** and **T2** with the same timing as address signals.

A refresh cycle outputs a 10 bit refresh address, on **MemAD2–11**. It is incremented after each refresh cycle.

5.6 Memory interface configuration

When **Reset** is high, the T414 is held in a reset state. When **Reset** goes low with **Analyse** held low, the memory controller executes the configuration sequence, selects the memory interface configuration, after which the T414 bootstraps.

During the configuration sequence, the memory interface controller performs some memory cycles. It also carries out eight refresh cycles, to initialize any dynamic RAM.

The **MemReq** signal is ignored until the configuration is complete.

The configuration sequence is described below. The complete sequence is always performed, but if the configuration disables refresh, the refresh cycles are dummy cycles which do not generate the control signals **notMemRd, notMemWrB**, or **notMemRf**.

The sequence is:

- The **Reset** signal goes low.
- A delay of 144 periods of the input clock **ClockIn**.
- The **MemAD** pins are scanned to see which (if any) of the preprogrammed interface configurations is selected. The scan lasts for 144 clock periods. During this period **notMemRd, notMemWrB0–3** and **notMemS0–4** are all held high.
- The memory interface performs 36 read cycles using the internal configuration associated with **MemAD31** to access the configuration (if any) in external ROM.
- A delay dependent on the selected configuration.
- Eight refresh cycles to initialize dynamic RAM.
- The T414 bootstraps.

5.6.1 Interface configuration selection

MemConfig determines the configuration to be selected. To select one of the 13 predefined configurations **MemConfig** is connected directly to one of the **MemAD** signals. The particular **MemAD** signal used determines the preset configuration.

If the configuration is to be supplied externally from ROM then **Memconfig** is driven via an inverter from one of the **MemAD** signals. When the interface performs the 36 configuration read cycles to access the external configuration, the data on that particular **MemAD** signal will be read as the configuration data.

Alternatively, the configuration can be supplied externally by special purpose logic (eg a PAL, small PROM or shift register). In this case the logic must hold **MemConfig** low during the address scan and must supply a 36 bit serial word to the **MemConfig** input during the 36 configuration read cycles. This must be inverse data.

The following table gives, for each preset configuration, the **MemConfig** connection, the duration of each Tstate, the width of **notMemS1**, and the delay from the start of **T2** to the falling edge of **notMems2** to **notMemS4**. These times are all in periods **Tm**. The table also gives the number of periods of the input clock between refresh cycles, e.g. 72 periods of 200ns give a refresh interval of 14.4 microseconds.

For each configuration the table gives the required memory access and cycle times (in multiples of processor cycles) plus the value of **e** to be used when estimating performance (see section 7.2.2 on T414 performance)

Pin	Duration of each Tstate periods Tm						Strobe duration				Write cycle type	Refresh interval input clocks	Cycle time proc cycles	Extra cycles e
	T1	T2	T3	T4	T5	T6	s1	s2	s3	s4				
MemAD0	1	1	1	1	1	1	30	1	3	5	late	72	3	2
MemAD1	1	2	1	1	1	2	30	1	2	7	late	72	4	3
MemAD2	1	2	1	1	2	3	30	1	2	7	late	72	5	4
MemAD3	2	3	1	1	2	3	30	1	3	8	late	72	6	5
MemAD4	1	1	1	1	1	1	3	1	2	3	early	72	3	2
MemAD5	1	1	2	1	2	1	5	1	2	3	early	72	4	3
MemAD6	2	1	2	1	3	1	6	1	2	3	early	72	5	4
MemAD7	2	2	2	1	3	2	7	1	3	4	early	72	6	5
MemAD8	1	1	1	1	1	1	30	1	2	3	early	–	3	2
MemAD9	1	1	2	1	2	1	30	2	5	9	early	–	4	3
MemAD10	2	2	2	2	4	2	30	2	3	8	late	72	7	6
MemAD11	3	3	3	3	3	3	30	2	4	13	late	72	9	8
MemAD31	4	4	4	4	4	4	31	30	30	18	late	72	12	11

To determine which configuration setting to use, the particular setup times, hold times, access times, and cycle times that the memory system requires must be calculated. A program is provided with the transputer development system for assisting in this task. The AC characteristics of all output signals are defined in terms of the appropriate number of periods **Tm**, plus or minus a maximum skew from the corresponding edge of **ProcClockOut**.

5.6.2 Externally supplied configuration

When supplying the configuration externally a sequence of 36 one bit values, as defined in the following table, must be presented on the **MemConfig** input. The resulting 36 bit word is effectively composed of 13 fields.

Fields 1 to 6 are 2 bit binary values which define the number of periods **Tm** that each Tstate is longer than its minimum time of one period **Tm**.

Field 7 is a 5 bit binary values defining the width of **notMemS1** in periods **Tm**.

Fields 8 to 10 are 5 bit binary values defining, respectively, the delay to the leading edges of **notMemS2–4** in periods **Tm**.

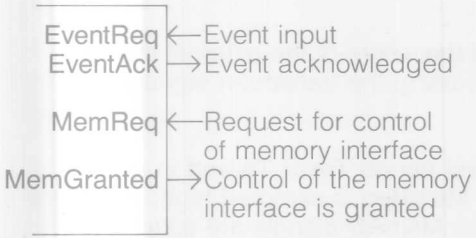
Field 11 is a 2 bit value which defines the refresh interval as ((Field11 + 1) * 18) periods of the input clock **ClockIn**, i.e., selecting one of the values 18, 36, 54, or 72.

If Field 12 = 1, refresh is enabled; otherwise no refresh cycles are generated.

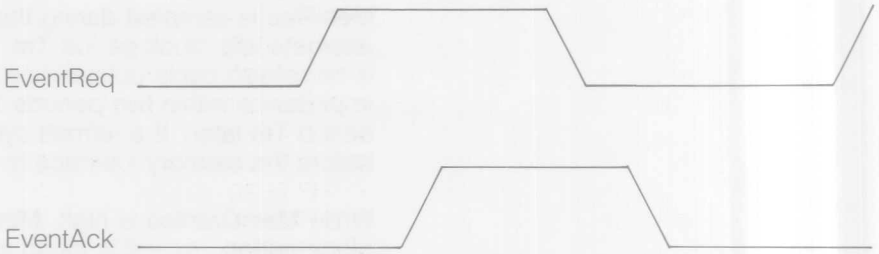
If Field 13 = 1, late write cycles are generated; otherwise early write cycles are generated.

The default setting of the configuration register used for initialization is the internal setting that would be selected if **MemConfig** were directly connected to **MemAD31**. This setting has each bit of the configuration register set to one. Thus, the slowest possible memory cycles are used (4 periods **Tm** for each of the Tstates **T1** to **T6**).

Value of MemAD during address phases of configuration interface cycles	Field	Function
#7FFFFF6C	1	T1 lsb
#7FFFFF70	1	T1 msb
#7FFFFF74	2	T2 lsb
#7FFFFF78	2	T2 msb
#7FFFFF7C	3	T3 lsb
#7FFFFF80	3	T3 msb
#7FFFFF84	4	T4 lsb
#7FFFFF88	4	T4 msb
#7FFFFF8C	5	T5 lsb
#7FFFFF90	5	T5 msb
#7FFFFF94	6	T6 lsb
#7FFFFF98	6	T6 msb
#7FFFFF9C	7	notMemS1 lsb
#7FFFFFA0	7	notMemS1
#7FFFFFA4	7	notMemS1
#7FFFFFA8	7	notMemS1
#7FFFFFAC	7	notMemS1 msb
#7FFFFFB0	8	notMemS2 lsb
#7FFFFFB4	8	notMemS2
#7FFFFFB8	8	notMemS2
#7FFFFFBC	8	notMemS2
#7FFFFFC0	8	notMemS2 msb
#7FFFFFC4	9	notMemS3 lsb
#7FFFFFC8	9	notMemS3
#7FFFFFCC	9	notMemS3
#7FFFFFD0	9	notMemS3
#7FFFFFD4	9	notMemS3 msb
#7FFFFFD8	10	notMemS4 lsb
#7FFFFFDC	10	notMemS4
#7FFFFFE0	10	notMemS4
#7FFFFFE4	10	notMemS4
#7FFFFFE8	10	notMemS4 msb
#7FFFFFEC	11	RefreshInterval lsb
#7FFFFFF0	11	RefreshInterval msb
#7FFFFFF4	12	RefreshEnable
#7FFFFFF8	13	LateWrite



The two event signals, **EventReq** and **EventAck**, together provide a handshaken interface with an occam process executing in the processor.



External logic takes **EventReq** high when the logic wishes to communicate with a process in the transputer. The rising edge of **EventReq** makes an external channel ready to communicate with the process. (This channel is additional to the external channels of the links.) When both the channel is ready and a process is ready to input from the channel, then the processor takes **EventAck** high and the process is scheduled. At any time after this point the external logic may take **EventReq** low, following which the processor will set **EventAck** low. After **EventAck** goes low, **EventReq** may go high to indicate the next event. Any further communication or synchronization necessary (for example, to tell external logic that the process has acted in response to the event) must be programmed explicitly.

If the process has high priority, and there is no other high priority process already running, then the maximum latency is 58 processor cycles, assuming that all memory accesses are to on–chip RAM. The typical latency is 12 cycles.

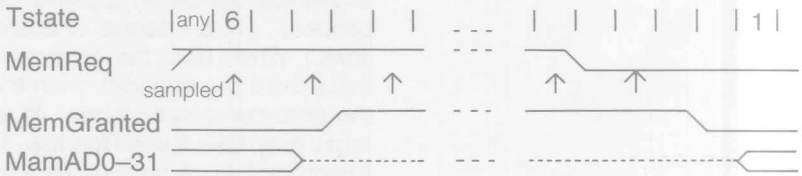
DMA via the memory interface

Peripheral controllers may access the whole of the external memory address space for DMA transfers, using the transputer signals **MemReq** and **MemGranted**.

MemReq is sampled during the last period **Tm** of **T6** and in every alternate idle clock period **Tm**. If **MemReq** is sampled high, and there is no refresh cycle outstanding, the **MemAD** signals are taken high impedance within two periods **Tm**. **MemGranted** is taken high one period **Tm** later. If a refresh cycle is outstanding, then it is completed before the memory interface is released.

While **MemGranted** is high, **MemReq** is sampled every alternate clock period Tm; if it is found to be low, **MemGranted** is taken low two periods **Tm** later, and any pending transputer interface cycle can start.

MemReq and MemGranted



The processor and links can continue accessing internal memory while DMA proceeds to external memory.

The performance of the transputer is measured in terms of the number of bytes to hold the program, and the number of (internal) processor cycles to execute it.

The figures here relate to occam programs. For the same function, other languages should achieve approximately the same performance as occam.

7.1 T414 speed selections

The following table illustrates the designation of the T414 speed selections.

Designation	Instruction throughput	Processor clock speed	Processor cycle time	Input clock frequency
IMS T414–12	6 MIPS	12.5 MHz	80 ns	5 MHz
IMS T414–15	7.5 MIPS	15 MHz	67 ns	5 MHz
IMS T414–20	10 MIPS	20 MHz	50 ns	5 MHz

7.2 Performance summary

These figures are averages obtained from detailed simulation, and should be used only as an initial guide. The following abbreviations are used to represent the quantities indicated

np	number of component processes
r	1 if INT or vector parameter, 0 if not
ts	number of table entries (table size)
w	width of constant in nibbles
p	number of places to shift
Eg	expression used in a guard
Et	timer expression used in a guard

	size bytes	time cycles
Identifiers		
variables		
in expression	1.1+r	2.1+2(r)
assigned to or input to	1.1+r	1.1+(r)
in PROC call, corresponding to an INT parameter	1.1+r	1.1+(r)
channels	1.1	2.1
Vector Variables		
constant subscript	0	0
variable subscript	5.3	7.3
expression subscript	5.3	7.3
Declarations		
CHAN	3.1	3.1
CHANsize	9.4	2.2 + 20.2*size
PROC	body+2	0
Primitives		
assignment	0	0
input	4	26.5
output	1	26
STOP	2	25
SKIP	0	0
Arithmetic operators		
+, -	1	1
*	2	39
/	2	40
REM	2	38
>>, <<	2	3+p
Boolean operators		
OR	4	8
AND, NOT	1	2

Comparison operators		
= constant	0	1
= variable	2	3
<> constant	1	3
<> variable	3	5
>, <	1	2
>=, <=	2	4
Bit operators		
BITAND, BITOR	2	2
><, BITNOT	2	2
Expressions		
constant in expression	w	w
check if error	4	6
Constructs		
SEQ	0	0
IF	1.3	1.4
if guard	3	4.3
ALT (non timer)	8	31
ALT (timer)	8	71
alt channel guard	10.2+2Eg	19.7+2Eg
timer alt guard	8+2Eg+2Et	41+2Eg+2AEt
skip alt guard	8+2Eg	10+2Eg
PAR	11.5+(np–1)*7.5	19.5+(np–1)*30.5
WHILE	4	12
Procedure call		
	3.5+(nparams–2)*1.1 +nvecparams*2.3	16.5+(nparams–2)*1.1 +nvecparams*2.3
Replicators		
replicated SEQ	7.3{+5.1}	–3.8+ 15.1*count{+7.1}
replicated IF	12.3{+5.1}	–2.6+ 19.4*count{+7.1}
replicated ALT	24.8{+10.2}	25.4+33.4*count{+14.2}
replicated timer ALT	24.8{+10.2}	62.4+33.4*count{+14.2}
replicated PAR	39.1{+5.1}	–6.4+ 70.9*count{+7.1}

{figures in curly brackets are not necessary if the number of replications is a compile time constant}

7.2.1 Floating point operations

Floating point operations are provided by a run–time package, which requires approximately 400 bytes of memory for the single length arithmetic operations, and 2500 bytes for the double length arithmetic operations. The following table summarizes the estimated performance of the package.

		Processor cycles (typical)	**Processor cycles** (worst)
REAL32	+, −	230	300
	*, /,	220	280
	<, >, =, >=, <=, <>	60	60
REAL64	+, −	480	550
	*, /,	490	600
	<, >, =, >=, <=, <>	60	60

7.2.2 Effect of external memory

Extra processor cycles may be needed when program and/or data are held in external memory, depending both on the operation being performed, and on the speed of the external memory. After a processor cycle which initiates a write to memory, the processor continues execution at full speed until at least the next memory access.

Whilst a reasonable estimate may be made of the effect of external memory, the actual performance will depend upon the exact nature of the given sequence of operations.

External memory is characterized by the number of extra processor cycles per external memory cycle, denoted as **e**. The value of **e** is 2 for the fastest external memory; a typical value for a large external memory is 5.

The number of additional cycles required to access data in external memory is **e**. If program is stored in external memory, and **e** has the value 2 or 3, then no extra cycles need be estimated for linear code sequences. For larger values of **e**, the number of extra cycles required for linear code sequences may be estimated at (**e** − 3)/4. A transfer of control may be estimated as requiring **e** + 3 cycles.

These estimates may be refined for various constructs. In the following table, **n** denotes the number of components in a construct. In the case of **IF**, the **n**'th conditional is the first to evaluate to **TRUE**, and the costs include the costs of the conditionals tested. The number of bytes in an array assignment or communication is denoted by **b**.

	Program off chip	Data off chip
Boolean expressions	e – 2	0
IF	3en – 8	en
Replicated **IF**	(6e – 4)n + 7	(5e – 2)n + 8
Replicated **SEQ**	(3e – 3)n + 2	(4e – 2)n
PAR	(3e – 1)n + 8	3en + 4
Replicated **PAR**	(10e – 8)n + 8	16en – 12
ALT	(2e – 4)n + 6e	(2e – 2)n + 10e – 8
array assignment and communication in one transputer	0	max (2e, e(b/2))

The effective rate of INMOS links is slowed down by external memory if all four links are operating in both directions simultaneously and **e** has a value of 6 or greater.

The following simulation results illustrate the effect of storing program and/or data in external memory. The results are normalized to 1 for both program and data on chip. The first program (Sieve of Erastosthenes) is an extreme case as it is dominated by small, data access intensive, loops; it contains no concurrency, communication, or even multiplication or division. The second program is the pipeline algorithm for Newton Raphson square root computation (given as a tutorial example in the occam programming manual).

	on chip	program off chip				data off chip				program and data off chip			
e		2	3	4	5	2	3	4	5	2	3	4	5
(1)	1	1.3	1.5	1.7	1.9	1.5	1.8	2.1	2.3	1.8	2.2	2.7	3.2
(2)	1	1.1	1.2	1.2	1.3	1.2	1.4	1.6	1.7	1.3	1.6	1.8	2.0

Note

Parameters given in this section will be revised as a result of fuller characterization.

8.1 Absolute maximum ratings

Parameter		Min	Max	Unit	Note
VCC	DC supply voltage	0	7.0	V	1, 2, 3
VI,VO	Input or output voltage on any pin	–0.5	VCC+0.5	V	1, 2, 3
OSCT	Output short circuit time (one pin)		1	S	1
TS	Storage temperature	–65	150	degC	1
TA	Ambient temperature under bias	–55	125	degC	1
PD	Power dissipation rating		2	W	

Notes

1 Stresses greater than those listed may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

2 All voltages are with respect to GND.

3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electric fields; however it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level such as GND.

8.2 Recommended operating conditions

Parameter		Min	Max	Unit	Note
VCC	DC supply voltage	4.5	5.5	V	1
VI,VO	Input or output voltage	0	VCC	V	1,2
II	Input current		+– 25	mA	3
CL	Load capacitance on any pin		50	pF	
CLMem	Total load capacitance on MemAD		1500	pF	
TA	Operating temperature range	0	70	C	

Notes

1 All voltages are with respect to GND.

2 Excursions beyond the supplies are permitted but not recommended; see DC characteristics.

3 The input current applies to any input or output pin and applies when the voltage on the pin is between GND and VCC.

8.3 DC characteristics

Parameter		Conditions	Min	Max	Unit	Note
VIH	High level input voltage		2.0	VCC+0.5	V	
VIL	Low level input voltage		–0.5	0.8	V	
II	Input current	GND < VI< VCC		+–200	uA	
VOH	Output high voltage	IOH = 2mA	VCC–1		V	
VOL	Output low voltage	IOL = 4mA		0.4	V	
IOS	Output short circuit current	GND < VO < VCC	*	*		2
IOZ	Tristate output current	GND < VI < VCC		+–200	uA	
PD	Power dissipation			0.9	W	
CIN	Input capacitance	f = 1 MHz		7	pF	
COZ	Output capacitance in tristate	f = 1 MHz	*		pF	2

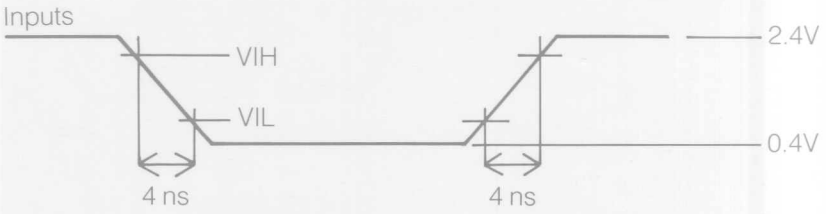
Notes

1 4.5 V < VCC < 5.5 V
0 degC < TA < 70 degC
Input clock frequency = 5 MHz
All voltages are with respect to GND

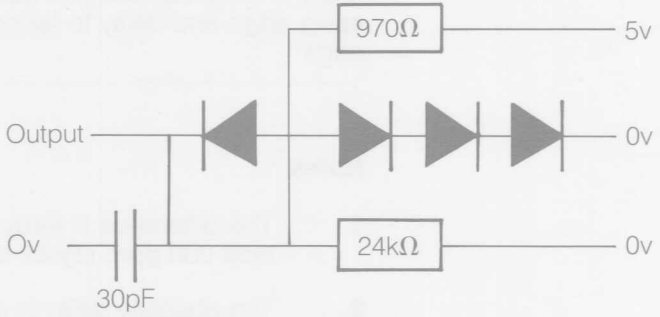
2 Entries marked with an asterisk (*) will be specified on the parameter list as a result of full characterization.

8.4 Measurement of AC characteristics

Reference points for AC characteristics



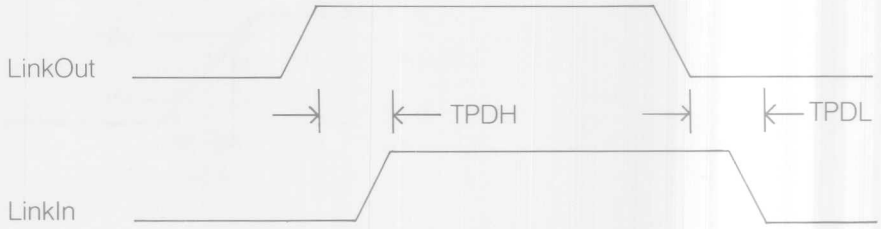
Load circuit for AC measurements



The load circuit approximates to two Schottky TTL loads, with total capacitance of 30 pF.

8.5 AC characteristics of INMOS serial links

INMOS serial link waveforms



Skew is the maximum difference between TPDH and TPDL

Parameter	Max	Unit	Note
Difference in frequency of input clock Clockin between two devices connected by a link	400	ppm	1
Skew in buffering between delay to rising edge and delay to falling edge	*	ns	2

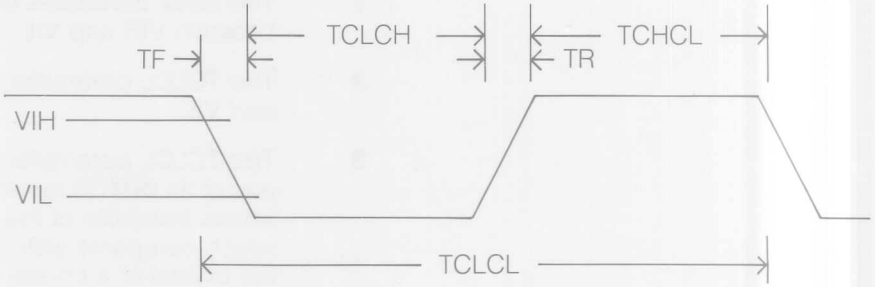
Notes

1 The difference in frequency of 400 ppm allows the use of low cost 200 ppm crystal oscillators.

2 The absolute delay in buffering between a LinkOut pin and a Linkin pin is immaterial because data reception is asynchronous. Waveform fidelity is important, however, and this is most easily expressed as a skew tolerance. The effect of skew is to narrow the pulse widths. The actual permissible skew will be specified on the parameter list as a result of full characterization.

8.6 AC characteristics of system services

ClockIn waveform



Parameter		Min	Max	Unit	Note
TCHCL	Clock pulse width high	40		ns	
TCLCH	Clock pulse width low	40		ns	
TR	Clock rise time from VIL to VIH	1.5	10	ns	
TF	Clock fall time from VIH to VIL	1.5	10	ns	1
TCLCL	Clock period	200	400	ns	2
delta TCLCL	Instability of clock period		+/– 250	ps	3
Cap	Electrolytic decoupling capacitor between CapPlus and CapMinus	8	20	uF	4
TRHRH	Reset pulse width high	8		ClockIn periods	
	Time VCC must be valid and ClockIn running before Reset goes low	10		ms	
	BootFromROM setup time before Reset or Analyse goes low	0			
	BootFromROM hold time after Reset goes low	50		ms	
	Analyse pulse width	8		ClockIn periods	

Notes

1 The clock transitions must be monotonic within the range between VIH and VIL.

2 The TCLCL parameter must be met at all voltages between VIH and VIL.

3 The TCLCL parameter is required to be met by any T414 using any of its INMOS serial links. The tolerance of +/– 250 ps allows instability of the electrical signal of 1250 ppm at 5 MHz, which compares with the 200 ppm recommended tolerance for the crystal of a crystal oscillator.

4 This specification is met by a 10 microFarad, +80% –20%, 10V capacitor.

8.7 Memory interface AC characteristics

The AC characteristics of the memory interface are dependent upon the speed of the transputer, the configuration of the memory interface, and the use of wait states.

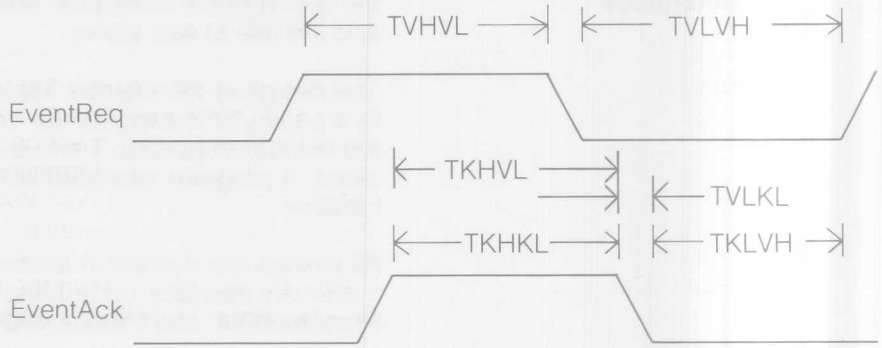
The design of the memory interface enables all the AC characteristics in a particular instance to be derived by a simple calculation involving the number of periods **Tm** that the signal is high or low, plus factors for skew. A program, provided with the development system, performs this function.

All timings are defined in relation to **ProcClockOut**. The start of a memory interface cycle always coincides with a rising edge of **ProcClockOut**, and **ProcClockOut** changes state at every period **Tm**.

Parameter	min	max	unit
Skew of any signal change to corresponding edge of **ProcClockOut**		+/– 3	ns
MemWait setup to falling edge of **ProcClockOut**	0.5**Tm**+*		ns
MemWait hold to falling edge of **ProcClockOut**	0.5**Tm**+*		ns
Read data setup	*		ns
Read data hold time	0		

8.8 Peripheral interfacing AC characteristics

Event signals waveforms



Parameter		Min	Max	Unit	Note
TVHVL	**EventReq** pulse width high	2		processor cycles	
TVLVH	**EventReq** pulse width low	2		processor cycles	
TVLKL	Falling edge delay from **EventReq** to **EventAck**	0	2	processor cycles	
TKHKL	**EventAck** pulse width high	2		processor cycles	
TKHVL	Delay from **EventAck** to falling edge of **EventReq**	0			
TKLVH	Delay from falling edge of **EventAck** to next **EventReq**	0			
	MemReq to **MemGrant** delay		2	memory cycles	
			+2	processor cycles	1
	MemAD tristate before**MemGrant**	0			
	Delay from **MemReq** going low to **MemGrant** low		2	processor cycles	

Notes

1 **MemReq** is sampled at the end of each memory cycle. If a refresh cycle is outstanding, the refresh cycle is completed before the memory interface is granted.

Full details of each signal are provided in the referenced section.

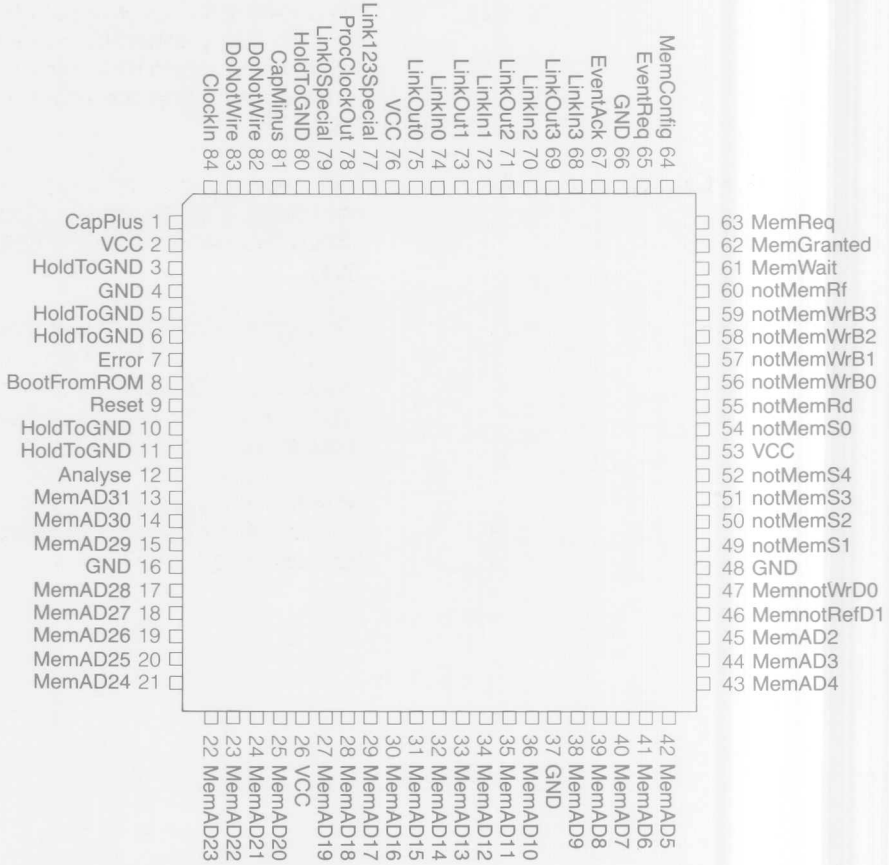
Signal	I/O	Function
Analyse	I	Signal used to investigate the state of a T414. The signal brings the processor, links and clocks to a halt within approx three timeslice periods (approx 2.5 milliseconds). **Reset** may then be applied, which will not destroy state information nor reinitialize the memory interface. After **Reset** has been taken low, **Analyse** may be taken low after which the processor will execute its bootstrap routine. (**2.3**, **3.4**)
BootFromROM	I	If this is held to **VCC**, then the boot program is taken from ROM. Otherwise the T414 awaits a bootstrap message from a link. (**3.3.1**)
CapMinus		The more negative terminal of an internal power supply used for the internal clocks. The signal must be connected to the negative terminal of a capacitor of nominal value 10 microfarads. (**8.6**)
CapPlus		This pin must be connected to the more positive terminal of the 10 microfarad capacitor connected to **CapMinus**.
ClockIn	I	Input clock from which all internal clocks are generated. The nominal input clock frequency for all transputers, of whatever wordlength and speed, is 5 MHz. (**8.6**)
DoNotWire	O	These are output pins reserved for INMOS use. They should be left unconnected.
Error	O	The processor has detected an error. Errors result from arithmetic overflow, by array bounds violations or by divide by zero. The **Error** flag can also be set by an instruction, to allow other forms of software detected error. (**2.3**)
EventAck	O	The **EventReq** and **EventAck** are a pair of handshake signals for external events. External logic takes **EventReq** high when the logic wishes to communicate with a process in the T414. The rising edge of **EventReq** causes a process to be scheduled. The processor takes **EventAck** high when the process is scheduled. At any time after this point the external logic may take **EventReq** low, following which the processor will set **EventAck** low. (**6**,**8.8**)
EventReq	I	Request external event. See description of **EventAck** above.

GND		Power supply return and logic reference, 0 V. There are several **GND** pins to minimize inductance within the package.
HoldToGND	I	These are input pins reserved for INMOS use. They should be held to ground either directly or through a resistor of less than 10K ohms.
LinkIn0 – 3	I	Input pins of the standard links. The **LinkIn** pin receives a serial stream of bits including protocol bits and data bits. Link inputs may be left floating or tied low. A link input must not be tied high. (**4**)
LinkOut0 – 3	O	Output pins of each of the standard links. A **LinkOut** pin may be left floating or may be connected to one (and only one) **LinkIn** pin. As long as the skew specification is met, the connection may be via buffers. (**4**)
Link0Special	I	Link 0 operates at the non–standard speed of 5 Mbit nominal if this pin is wired high. (**4.1.1**)
Link123Special	I	Links 1, 2 and 3 operate at the non–standard speed of 5 Mbit nominal if this pin is wired high. (**4.1.1**)
MemAD	I/O	32–bit wide multiplexed external memory address and data bus. (**5**)
MemConfig	I	The **MemConfig** input determines how the external memory interface is to be configured. It may be connected to any of the **MemAd** pins directly, or to the output of an inverter which takes its input from any of the **MemAD** pins in order to select the required memory configuration. (**5.6**)
MemGranted	O	See description below of **MemReq**.
MemReq	I	This input is used by external devices to access the memory interface **MemAD**. When **MemReq** is sampled high, and if there is no refresh cycle outstanding, the **MemAD** signals are taken high impedance. **MemGranted** is taken high one period **Tm** later. If a refresh cycle is outstanding when **MemReq** is sampled, the refresh cycle is completed before access to the memory interface is granted. While **MemGranted** is high, **MemReq** is sampled every alternate clock period **Tm**; if it is found to be low, **MemGranted** is taken low two periods **Tm** later, and any pending transputer interface cycle can start. (**6**)

MemWait	I	Wait input for the memory interface. The **MemWait** signal is sampled synchronously just before the end of **T4**. If, at the time **MemWait** is sampled, the input is low, the interface cycle proceeds. Otherwise, the interface is held in **T4** until the input is sampled and found to be low; one period **Tm** later the interface cycle proceeds to **T5**. (**5.4**)
notMemRd	O	Used for Read Data, Read Enable, Output Enable for external memory. The **notMemRd** signal is taken low during read cycles, including those which occur during configuration, at the start of **T4**, and is taken high at the start of **T6**. (**5.2**)
notMemRf	O	Refresh indicator. This signal goes low at the start of **T1** if the current cycle is a refresh. It goes high one period **Tm** after the start of **T6**. (**5.5**)
notMemS0	O	AddressValid strobe, LatchEnable, or ChipEnable for external memory. The **notMemS0** signal goes low at the start of **T2** and goes high at the start of **T6**. (**5**)
notMemS1	O	External memory strobe with configurable width, normally used for RowAddressStrobe, AddressLatchEnable, or ChipEnable. (**5**)
notMemS2 – 4	O	Three separate external memory strobes with configurable delay. Their falling edge can be configured to go low between zero and 31 periods **Tm** after the start of **T2**, and they go high at the start of **T6**. The conventional use suggested for the strobes is: **notMemS2** use as AddressMultiplex for dynamic RAMs **notMemS3** use as CAS for dynamic RAMs **notMemS4** use as wait state generator If the system does not use dynamic RAMs, **notMemS2** and **notMemS3** may be used as wait state generators with different delays from **notMemS4**. (**5**)

notMemWrB0 – 3	O	Separate write strobe for each byte of external memory. The **notMemWrB** signal for each byte only goes low if the relevant byte is to be written. The write strobes may be configured to be either early or late. Early write strobes allow a wide write pulse to static RAM and allow common I/O for dynamic RAM. Late writes allow the data to be strobed by either the falling edge or the rising edge of **notMemWrB**. (**5.3**)
ProcClockOut	O	The processor clock which is output in phase with the memory interface. The processor clock frequency is a multiple of the input clock frequency. The multiple differs for different speed parts. (**5.4**, **7.1**)
Reset	I	The falling edge of **Reset** initializes the T414, triggers the sequence which configures the memory interface and then starts the processor executing a bootstrap routine. If **Analyse** is asserted during reset, then the memory interface is not reconfigured, and sufficient state is preserved to permit software analysis. (**3.1**)
VCC		Power supply, nominally 5 V. There are several **VCC** pins to minimize inductance within the package. **VCC** should be decoupled to **GND** by at least one 100 nF low inductance (such as ceramic) capacitor.

The T414 is available in an 84 pin chip J–Lead chip carrier.



inmos®

Product data

IMS C001 link adaptor

72 TRN 051 01

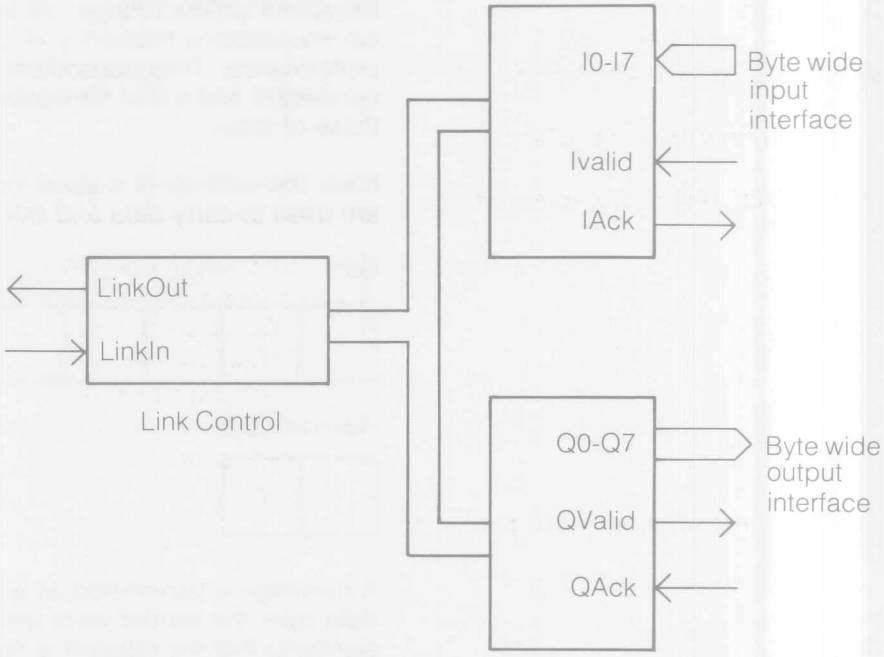
The IMS C001 link adaptor provides interfacing capabilities for transputers. Full details of the INMOS transputer architecture are given in the transputer architecture manual.

This manual details the product specific aspects of the IMS C001 and contains all the data necessary to engineer the device.

Other information relevant to all transputer products is contained in the occam programming manual (supplied with INMOS software products and available as a separate publication), and the transputer development system manual (supplied with the development system).

This is the first edition of the manual, dated September 1985.

Contents

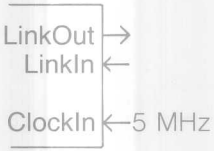


The IMS C001 link adaptor converts between an INMOS serial link and two fully handshaken byte–wide interfaces. One interface is for data coming from the serial link and one for data going to the serial link.

The serial link enables the link adaptor to communicate with another link adaptor, a transputer or an INMOS peripheral processor. Data reception is asynchronous which allows communication to be independent of clock phase. Transfers may proceed in both directions at the same time.

The IMS C001 provides programmable I/O pins for a transputer. Two link adaptors, directly connected via their byte–wide interfaces and inserted into an INMOS serial link, will correctly maintain the handshaken link protocol. This allows communication between transputers running at different clock frequencies.

Two link adaptors, connected via their links, can provide a high performance handshaken serial link between microprocessors from different families.



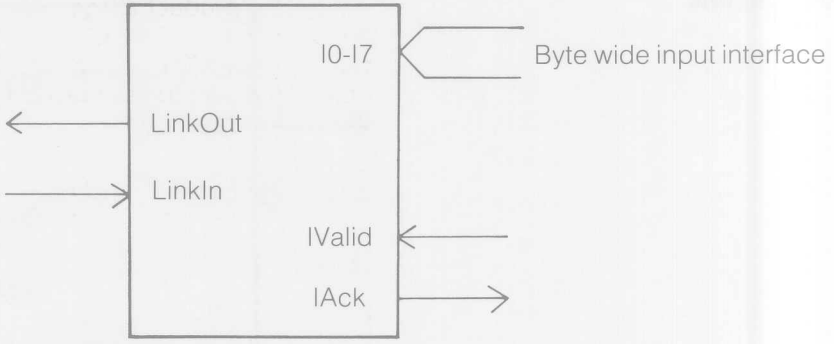
The INMOS serial links are a standard across all products in the transputer product range. All transputers will support a standard communications frequency of 10M bits/sec, regardless of processor performance. Thus transputers of different performance can be directly connected and future transputer systems will directly communicate with those of today.

Each link consists of a serial input and a serial output, both of which are used to carry data and link control information.

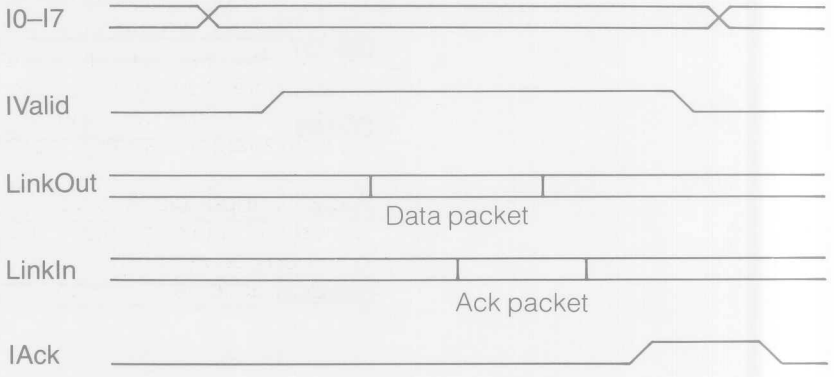
A message is transmitted as a sequence of bytes. After transmitting a data byte, the sender waits until an acknowledge has been received, signifying that the receiver is ready to receive another byte. The receiver can transmit an acknowledge as soon as it starts to receive a data byte, so that transmission can be continuous. This protocol synchronizes the communication of each byte of data, ensuring that slow and fast transputers communicate reliably.

A 5 MHz input clock is used, from which internal timings are generated. Link communication is not sensitive to clock phase. Thus, communication can be achieved between independently clocked systems as long as the communications frequency is the same.

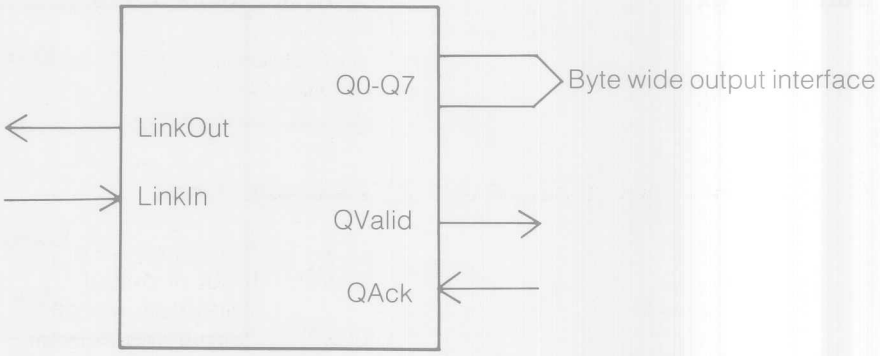
Output to link



IValid and **IAck** provide a simple two wire handshake. Data is presented to the link adaptor on **I0–I7**, and **IValid** is taken high to commence the handshake. The adaptor transmits the data through the serial link, and acknowledges receipt of the data on **IAck** when it has received the acknowledgment from the serial link and all the data bits have been transmitted. **IValid** is then taken low, and the link adaptor completes the handshake by taking **IAck** low.



Input from link



The link adaptor receives data from the serial link, presents it on **Q0– Q7**, and takes **QValid** high to commence the handshake. Receipt of the data is acknowledged on **QAck**, and the link adaptor then transmits an acknowledgement on the serial link. The link adaptor takes **QValid** low and the handshake is completed by taking **QAck** low.

Note

Parameters given in this section will be revised as a result of fuller characterization.

4.1 Absolute maximum ratings

Parameter		Min	Max	Unit	Note
VCC	DC supply voltage	0	7.0	V	1, 2, 3
VI,VO	Input or output voltage on any pin	–0.5	VCC+0.5	V	1, 2, 3
OSCT	Output short circuit time (one pin)		1	S	1
TS	Storage temperature	–65	150	degC	1
TA	Ambient temperature under bias	–55	125	degC	1
PD	Power dissipation rating		500	mW	

Notes

1 Stresses greater than those listed may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

2 All voltages are with respect to GND.

3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electric fields; however it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level such as GND.

4.2 Recommended operating conditions

Parameter		Min	Max	Unit	Note
VCC	DC supply voltage	4.5	5.5	V	1
VI,VO	Input or output voltage	0	VCC	V	1,2
II	Input current		+– 25	mA	3
CL	Load capacitance on any pin		50	pF	
TA	Operating temperature range	0	70	C	

Notes

1 All voltages are with respect to **GND**.

2 Excursions beyond the supplies are permitted but not recommended; see DC characteristics.

3 The input current applies to any input or output pin and applies when the voltage on the pin is between **GND** and **VCC**.

4.3 DC characteristics

Parameter		Conditions	Min	Max	Unit	Note
VIH	High level input voltage		2.0	VCC+0.5	V	
VIL	Low level input voltage		–0.5	0.8	V	
II	Input current	GND < VI< VCC		+–200	uA	
VOH	Output high voltage	IOH = –1.5mA	VCC–1		V	
VOL	Output low voltage	IOL = 3mA		0.4	V	
IOS	Output short circuit current	GND < VO < VCC	*	*		2
IOZ	Tristate output current	GND < VI < VCC		+–200	uA	
PD	Power dissipation			40	mW	
CIN	Input capacitance	f = 1 MHz		7	pF	
COZ	Output capacitance in tristate	f = 1 MHz	*		pF	2

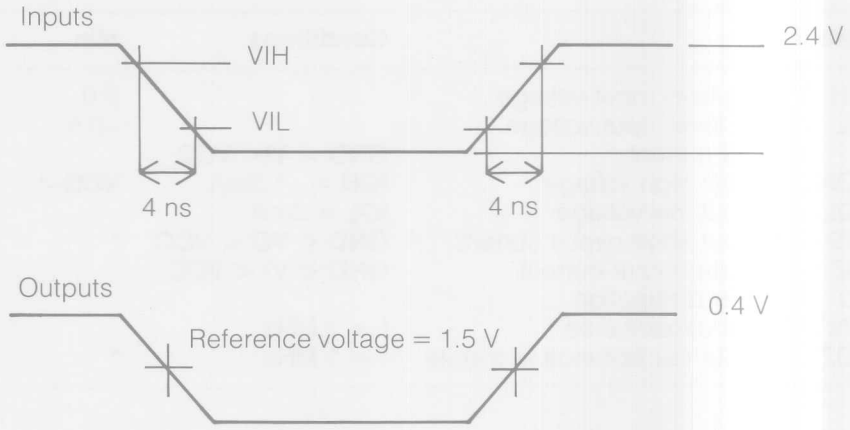
Notes

1 4.5 V < VCC < 5.5 V
0 degC < TA < 70 degC
Input clock frequency = 5 MHz
All voltages are with respect to GND

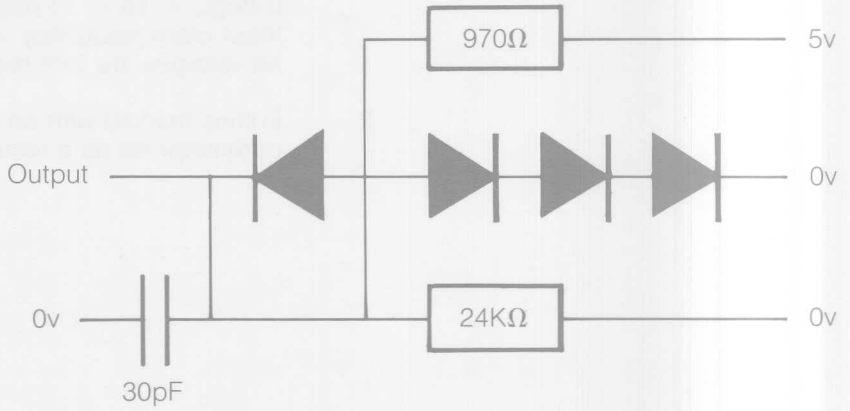
2 Entries marked with an asterisk (*) will be specified on the parameter list as a result of full characterization.

4.4 Measurement of AC characteristics

Reference points for AC characteristics



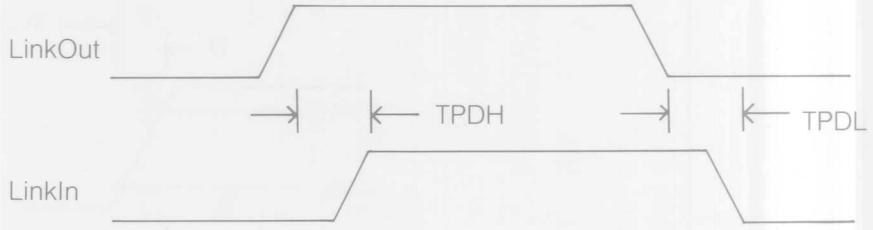
Load circuit for AC measurements



The load circuit approximates to two TTL loads, with a total capacitance of 30 pF.

4.5 AC characteristics of INMOS serial links

INMOS serial link waveforms



Skew is the maximum difference between TPDH and TPDL

Parameter	Max	Unit	Note
Difference in frequency of input clock **Clockin** between two devices connected by a link	400	ppm	1
Skew in buffering between delay to rising edge and delay to falling edge	*	ns	2

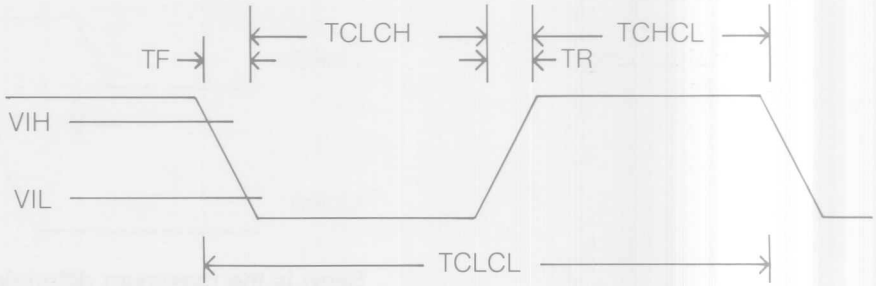
Notes

1 The difference in frequency of 400 ppm allows the use of low cost 200 ppm crystal oscillators.

2 The absolute delay in buffering between a **LinkOut** pin and a **Linkin** pin is immaterial because data reception is asynchronous. Waveform fidelity is important, however, and this is most easily expressed as a skew tolerance. The effect of skew is to narrow the pulse widths. The actual permissible skew will be specified on the parameter list as a result of full characterization.

4.6 AC characteristics of system services

ClockIn waveform



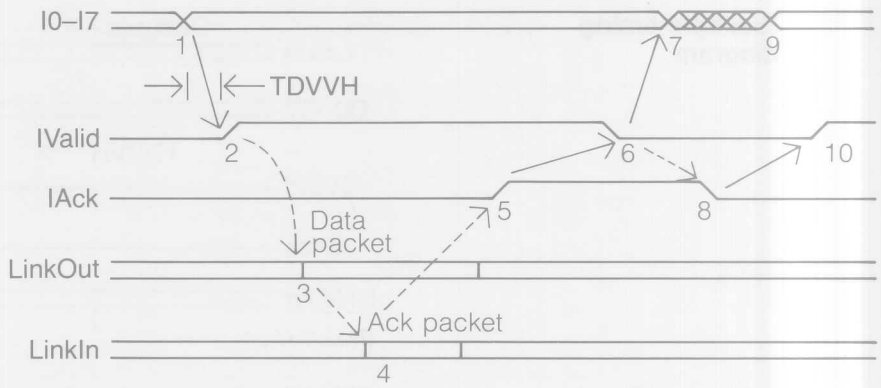
Parameter		Min	Max	Unit	Note
TCHCL	Clock pulse width high	40		ns	
TCLCH	Clock pulse width low	40		ns	
TR	Clock rise time from VIL to VIH	1.5	10	ns	
TF	Clock fall time from VIH to VIL	1.5	10	ns	1
TCLCL	Clock period	200	400	ns	2
delta TCLCL	Instability of clock period		+/– 250	ps	3
Cap	Electrolytic decoupling capacitor between CapPlus and GND	8	20	uF	4
TRHRH	Reset pulse width high	8		ClockIn periods	

Notes

1. The clock transitions must be monotonic within the range between VIH and VIL.
2. The TCLCL parameter must be met at all voltages between VIH and VIL.
3. The tolerance of +/– 250 ps allows instability of the electrical signal of 1250 ppm at 5 MHz, which compares with the 200 ppm recommended tolerance for the crystal of a crystal oscillator.
4. This specification is met by a 10 microFarad, +80% –20%, 10V capacitor.

4.7 AC characteristics of parallel interface

Output to link



Example timing diagram

Event details

1 Data is presented on **I0–7**
2 The presence of valid data is indicated by raising **IValid**
3 The C001 commences transmitting the data on **LinkOut**
4 The C001 receives an acknowledge packet on **LinkIn**
5 The C001 acknowledges the data by taking **IAck** high
6 **IValid** is taken low
7 Data is removed from **I0–7**
8 The C001 completes the handshake by taking **IAck** low
9 Another data byte is presented on **I0–7**
10 The presence of valid data is indicated by raising **IValid**

Dependencies

Ensure that 1 precedes 2 by **TDVVH**
5 precedes 6
6 precedes 7
8 precedes 10

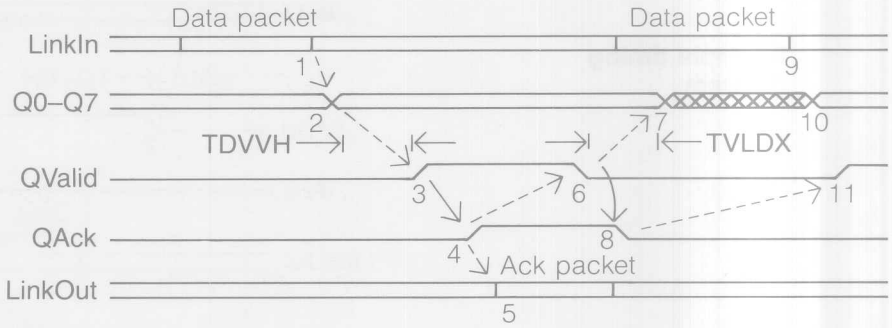
The C001 ensures that 2 precedes 3
4 precedes 5
6 precedes 8

The link protocol ensures that 3 precedes 4

Parameter		Min	Max	Unit
TDVVH	Data setup before **IValid** high	5		ns

Input from link

Example timing diagram



Event details

1 The C001 receives on **LinkIn** the data packet to be output
2 The C001 places the data on **Q0–7**
3 The C001 indicates the presence of valid data by raising **QValid**
4 The data is acknowledged by taking **QAck** high
5 The C001 transmits an acknowledge packet on **LinkOut**
6 The C001 takes **QValid** low
7 The C001 no longer holds the data outputs **Q0–7** valid
8 The handshake is completed by taking **QAck** low
9 The C001 receives on **LinkIn** another data packet
10 The C001 places the data on **Q0–7**
11 The C001 indicates the presence of valid data by raising **QValid**

Dependencies

Ensure that
3 precedes 4
6 precedes 8

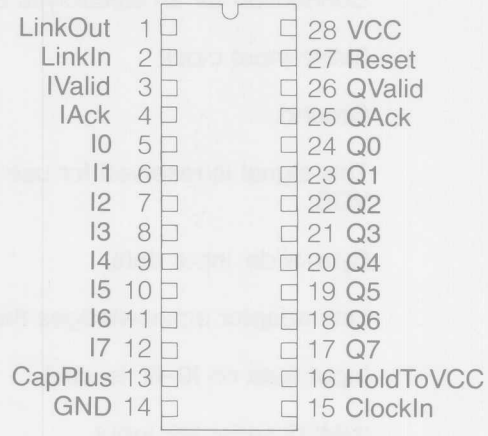
The C001 ensures that
1 precedes 2
2 precedes 3 by **TDVVH**
4 precedes 5
4 precedes 6
6 precedes 7 by **TVLDX**
8 precedes 11

Parameter		Min	Max	Unit
TDVVH	Data setup before **QValid** high	15		ns
TVLDX	Data hold after **QValid** low	15		ns

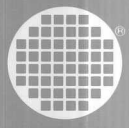
Signal	I/O	Description
CapPlus		Connection for an electrolytic capacitor.
ClockIn	I	5MHz input clock.
GND	I	Ground.
HoldToVCC		This signal is reserved for use by INMOS, and must be connected to **VCC**.
I0–I7	I	Byte–wide input data.
IAck	O	Link adaptor acknowledges that input data has been received.
IValid	I	Input data on **I0–I7** is valid.
LinkIn	I	INMOS serial link input.
LinkOut	O	INMOS serial link output.
Q0–Q7	O	Byte–wide output data.
QAck	I	Acknowledge to link adaptor that data output from the link adaptor has been received.
QValid	O	Output data on **Q0–Q7** is valid.
Reset	I	Reset link adaptor.
VCC	I	+5 volt supply input.

Package

The C001 is available in a 600 mil 28 pin DIP package.



inmos®



Product data

IMS C002 link adaptor

72 TRN 052 01

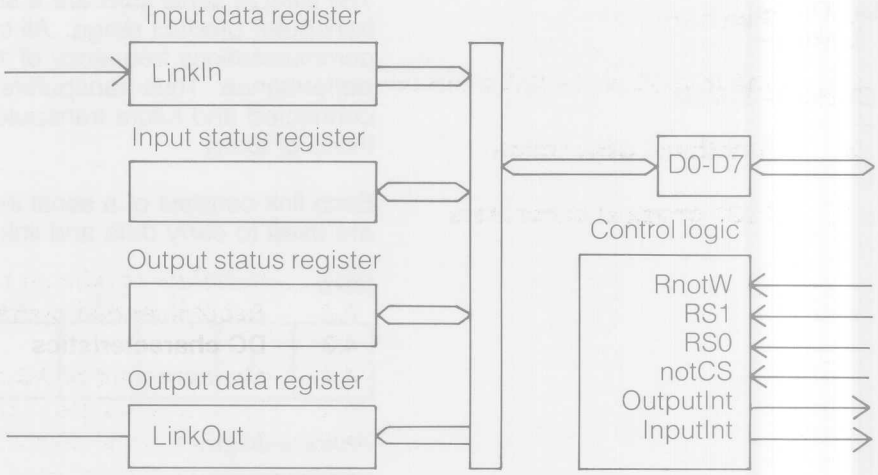
The IMS C002 link adaptor provides interfacing capabilities for transputers. Full details of the INMOS transputer architecture are given in the transputer architecture manual.

This manual details the product specific aspects of the IMS C002 and contains all the data necessary to engineer the device.

Other information relevant to all transputer products is contained in the occam programming manual (supplied with INMOS software products and available as a separate publication), and the transputer development system manual (supplied with the development system).

This is the first edition of the manual, dated September 1985.

Contents

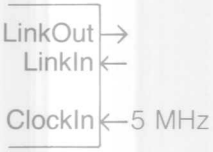


The IMS C002 link adaptor provides an interface between an INMOS serial link and a microprocessor system bus, via an 8–bit bi–directional interface.

The IMS C002 has status/control and data registers for both input and output. Any of these can be accessed at the byte wide interface at any time. Two interrupt lines are provided, each gated by an interrupt enable flag. One presents an interrupt on output ready, and the other on data present.

The serial link enables the link adaptor to communicate with another link adaptor, a transputer or an INMOS peripheral processor. Data reception is asynchronous which allows communication to be independent of clock phase. Transfers may proceed in both directions at the same time.

The IMS C002 enables transputers to communicate with peripherals designed for conventional microprocessor systems. The link adaptor also makes it possible to build arrays with conventional microprocessors, and to link microprocessors from different families.



The INMOS serial links are a standard across all products in the transputer product range. All transputers will support a standard communications frequency of 10M bits/sec, regardless of processor performance. Thus transputers of different performance can be directly connected and future transputer systems will directly communicate with those of today.

Each link consists of a serial input and a serial output, both of which are used to carry data and link control information.

A message is transmitted as a sequence of bytes. After transmitting a data byte, the sender waits until an acknowledge has been received, signifying that the receiver is ready to receive another byte. The receiver can transmit an acknowledge as soon as it starts to receive a data byte, so that transmission can be continuous. This protocol synchronizes the communication of each byte of data, ensuring that slow and fast transputers communicate reliably.

A 5 MHz input clock is used, from which internal timings are generated. Link communication is not sensitive to clock phase. Thus, communication can be achieved between independently clocked systems as long as the communications frequency is the same.

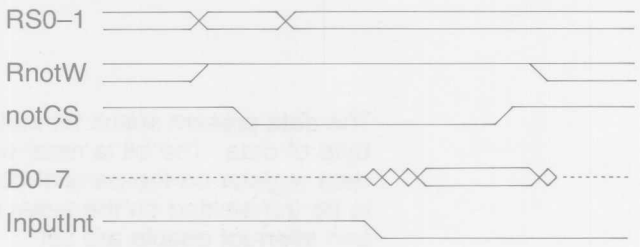
The C002 link adaptor is controlled at the parallel interface by reading and writing status/control registers, and by reading and writing data registers. Two interrupt lines are provided. One indicates that the link adaptor is ready to output a byte to the link, and the other indicates that it is holding a byte which it has read from the link.

Parallel interface

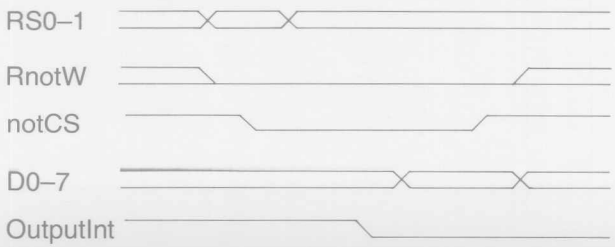
One of the four registers is selected by **RS0** and **RS1**. If a new value is to be written into the selected registers, it is set up on **D0–D7** and **RnotW** is taken low. **notCS** is then taken low. On read cycles, the C002 places the current value of the selected register on **D0–D7**.

RS1	RS0	RnotW	**Function**
0	0	1	**D0–D7** := input data register
0	1	0	output data register := **D0–D7**
1	0	1	**D0–D7** := input status register
1	0	0	input status register := **D0–D7**
1	1	1	**D0–D7** := output status register
1	1	0	output status register := **D0–D7**

Read register timing diagram

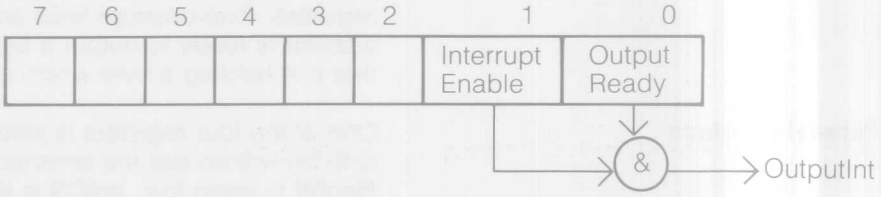


Write register timing diagram



Output to link

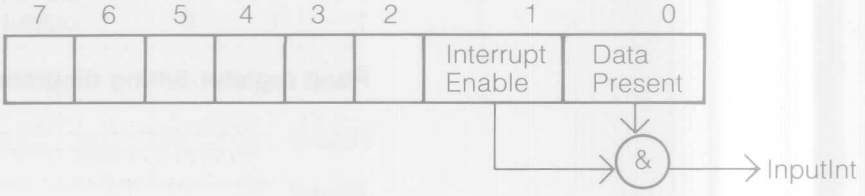
Output status register



The **output ready** status bit indicates that the serial link is able to receive a byte of data. The bit is set on reset, and when the C002 receives an acknowledgement from the serial link. It is reset when a data byte is written to the output data register on the parallel interface. **OutputInt** is set if both **output ready** and **interrupt enable** are set.

Input from link

Input status register



The **data present** status bit indicates that the serial link has received a byte of data. The bit is reset when the data byte is read from the input data register on the parallel interface, this causes an acknowledgement to be transmitted on the serial link. **InputInt** is set if both **data present** and **interrupt enable** are set.

Note

Parameters given in this section will be revised as a result of fuller characterization.

4.1 Absolute maximum ratings

Parameter		Min	Max	Unit	Note
VCC	DC supply voltage	0	7.0	V	1, 2, 3
VI,VO	Input or output voltage on any pin	–0.5	VCC+0.5	V	1, 2, 3
OSCT	Output short circuit time (one pin)		1	S	1
TS	Storage temperature	–65	150	degC	1
TA	Ambient temperature under bias	–55	125	degC	1
PD	Power dissipation rating		500	mW	

Notes

1 Stresses greater than those listed may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

2 All voltages are with respect to GND.

3 This device contains circuitry to protect the inputs against damage caused by high static voltages or electric fields; however it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level such as GND.

4.2 Recommended operating conditions

Parameter		Min	Max	Unit	Note
VCC	DC supply voltage	4.5	5.5	V	1
VI,VO	Input or output voltage	0	VCC	V	1,2
II	Input current		+– 25	mA	3
CL	Load capacitance on any pin		50	pF	
TA	Operating temperature range	0	70	C	

Notes

1 All voltages are with respect to **GND**.

2 Excursions beyond the supplies are permitted but not recommended; see DC characteristics.

3 The input current applies to any input or output pin and applies when the voltage on the pin is between **GND** and **VCC**.

4.3 DC characteristics

Parameter		Conditions	Min	Max	Unit	Note
VIH	High level input voltage		2.0	VCC+0.5	V	
VIL	Low level input voltage		–0.5	0.8	V	
II	Input current	GND < VI< VCC		+–200	uA	
VOH	Output high voltage	IOH = –1.5mA	VCC–1		V	
VOL	Output low voltage	IOL = 3mA		0.4	V	
IOS	Output short circuit current	GND < VO < VCC	*	*		2
IOZ	Tristate output current	GND < VI < VCC		+–200	uA	
PD	Power dissipation			40	mW	
CIN	Input capacitance	f = 1 MHz		7	pF	
COZ	Output capacitance in tristate	f = 1 MHz	*		pF	2

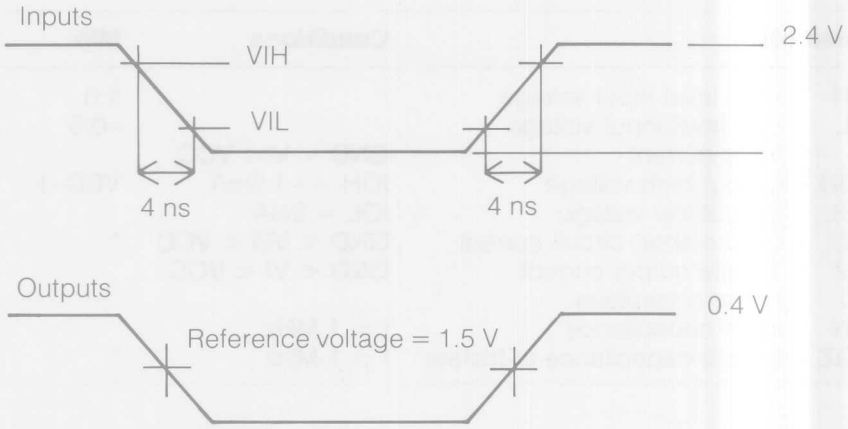
Notes

1 4.5 V < VCC < 5.5 V
0 degC < TA < 70 degC
Input clock frequency = 5 MHz
All voltages are with respect to GND

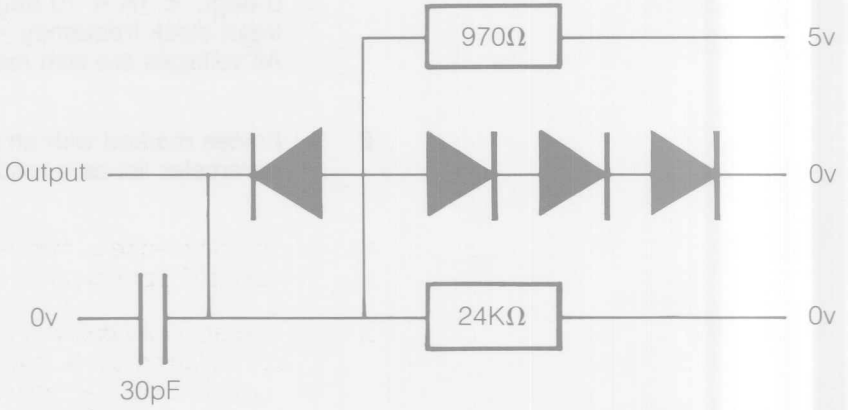
2 Entries marked with an asterisk (*) will be specified on the parameter list as a result of full characterization.

4.4 Measurement of AC characteristics

Reference points for AC characteristics



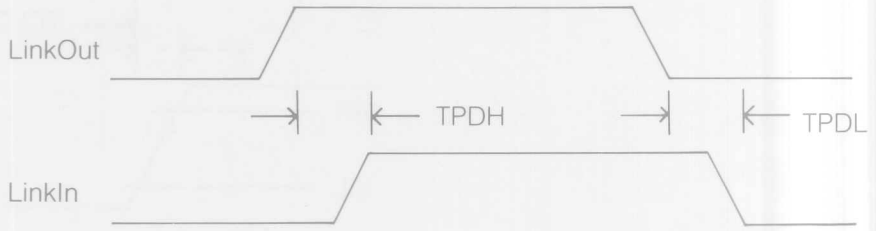
Load circuit for AC measurements



The load circuit approximates to two TTL loads, with a total capacitance of 30 pF.

4.5 AC characteristics of INMOS serial links

INMOS serial link waveforms



Parameter	Max	Unit	Note
Difference in frequency of input clock **Clockin** between two devices connected by a link	400	ppm	1
Skew in buffering between delay to rising edge and delay to falling edge	*	ns	2

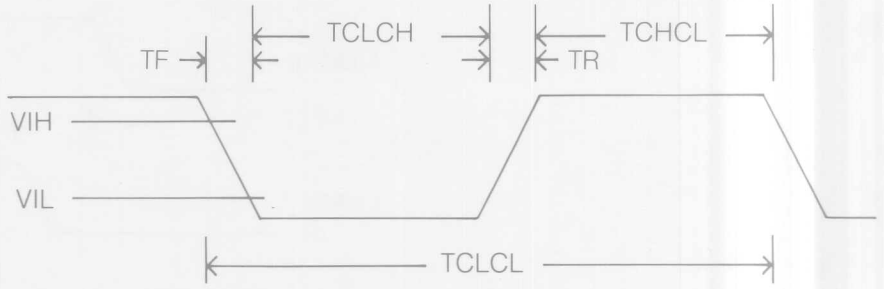
Notes

1 The difference in frequency of 400 ppm allows the use of low cost 200 ppm crystal oscillators.

2 The absolute delay in buffering between a **LinkOut** pin and a **Linkin** pin is immaterial because data reception is asynchronous. Waveform fidelity is important, however, and this is most easily expressed as a skew tolerance. The effect of skew is to narrow the pulse widths. The actual permissible skew will be specified on the parameter list as a result of full characterization.

4.6 AC characteristics of system services

ClockIn waveform



Parameter		Min	Max	Unit	Note
TCHCL	Clock pulse width high	40		ns	
TCLCH	Clock pulse width low	40		ns	
TR	Clock rise time from VIL to VIH	1.5	10	ns	
TF	Clock fall time from VIH to VIL	1.5	10	ns	1
TCLCL	Clock period	200	400	ns	2
delta TCLCL	Instability of clock period		+/– 250	ps	3
Cap	Electrolytic decoupling capacitor between CapPlus and GND	8	20	uF	4
TRHRH	Reset pulse width high	8		ClockIn periods	

Notes

1 The clock transitions must be monotonic within the range between VIH and VIL.

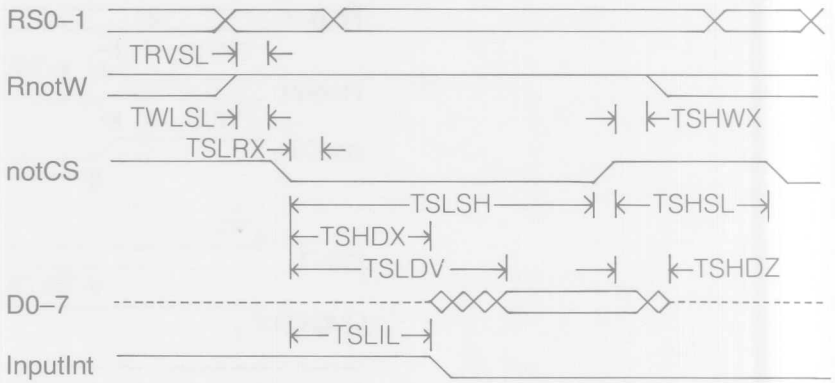
2 The TCLCL parameter must be met at all voltages between VIH and VIL.

3 The tolerance of +/– 250 ps allows instability of the electrical signal of 1250 ppm at 5 MHz, which compares with the 200 ppm recommended tolerance for the crystal of a crystal oscillator.

4 This specification is met by a 10 microFarad, +80% –20%, 10V capacitor.

4.7 AC characteristics of parallel interface

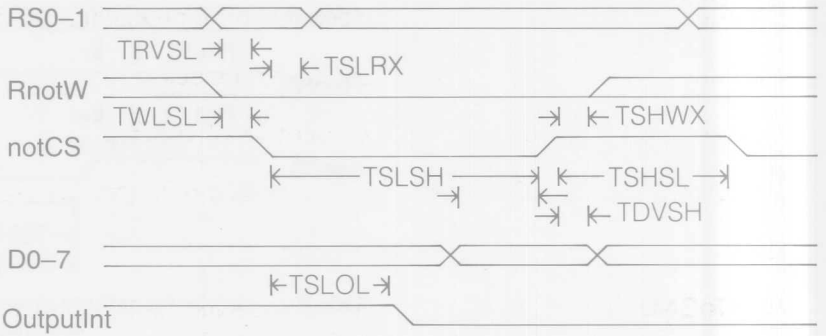
Read cycle



Parameter		Min	Max	Unit
TRVSL	Register select set–up to start of read	5		ns
TWHSL	**RnotW** set–up to start of read	5		ns
TSLRX	Register select hold after start of read	5		ns
TSLSH	Read pulse width	50		ns
TSLDX	Chip select to output active	5		ns
TSLDV	Chip select to data valid		40	ns
TSHWX	**RnotW** hold after end of read	5		ns
TSHDZ	Chip disable to output disable	0	25	ns
TSLIL	**InputInt** disabled after read		25	ns
TSHSL	Time between cycles	50		ns

4.7 AC characteristics of parallel interface

Write cycle



Parameter		Min	Max	Unit
TRVSL	Register select set–up to start of write	5		ns
TWLSL	**RnotW** set–up to start of write	5		ns
TSLRX	Register select hold after start of write	5		ns
TSHWX	**RnotW** hold after end of write	5		ns
TSLSH	Write pulse width	50		ns
TDVSH	Data set–up to end of write	15		ns
TSHDX	Data hold after end of write	5		ns
TSLOL	**OutputInt** disabled after write		25	ns
TSHSL	Time between cycles	50		ns

Signal	I/O	Description
CapPlus		Connection for an electrolytic capacitor.
ClockIn	I	5MHz input clock.
D0–D7	I/O	Bi–directional databus.
DoNotWire	O	Signal reserved for INMOS use. Must be left unconnected.
GND		Ground.
HoldToGND	I	Signal reserved for INMOS use. Must be held to **GND**. Failure to do so may cause damage to the device.
InputInt	O	Input Interrupt.
LinkIn	I	INMOS serial link input.
LinkOut	O	INMOS serial link output.
notCS	I	Chip select input.
OutputInt	O	Output Interrupt.
Reset	I	Reset link adaptor.
RnotW	I	Read or write register.
RS0–RS1	I	Register select lines; used in conjunction with **notCS** and **RnotW** to control the selection of the internal register to be read or written.
VCC		+5 volt supply input.

Package

The C002 is available in a 300 mil 24 pin DIP package.